

Systems Reference Library

IBM DOS Full American National Standard COBOL

Program Numbers: (Versions 1 & 2) 360N-CB-482

(Version 3) 5736-CB2 (Compiler Only)

5736-LM2 (Library Only)

DOS/VS COBOL 5746-CB1 (Compiler & Library)

5746-LM4 (Library Only)

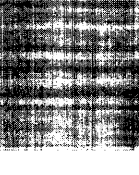
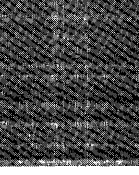
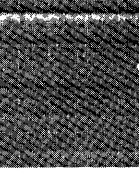
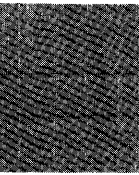
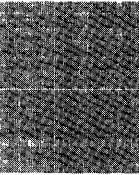
This publication gives the programmer the rules for writing programs that are to be compiled by the IBM DOS/VS COBOL and IBM Full American National Standard COBOL compilers under the Disk Operating System. It is meant to be used as a reference manual in the writing of IBM Full American National Standard COBOL programs.

COBOL (Common Business Oriented Language) is a programming language, similar to English, that is used for commercial data processing. It was developed by the Conference On Data Systems Languages (CODASYL). The Standard of the language is American National Standard COBOL, X3.23-1968, as approved by the American National Standards Institute (ANSI). American National Standard COBOL is compatible with, and identical to international standard ISO/R1989-1972, Programming Language COBOL.

IBM DOS/VS COBOL and IBM DOS Full American National Standard COBOL, Version 3, which include all the features of earlier versions, incorporate the eight processing modules defined in the highest level of the American national standard. These modules include:

- Nucleus
- Table Handling
- Sequential Access
- Random Access
- Sort
- Report Writer
- Segmentation
- Library

A significant number of IBM extensions are implemented as well; these extensions are printed on a shaded background.



PREFACE

This publication describes the IBM implementation of Full American National Standard COBOL, and all IBM extensions to that standard. Some statements are extensions to either American National Standard COBOL or to both American National Standard COBOL and the complete definition of CODASYL COBOL.

COBOL rules and allow for greater programming convenience.

All such extensions are printed on a shaded background for the convenience of users who wish strict conformance with the standard. Use of features that are extensions to the standard may result in incompatibilities between the implementation represented by this document and other implementations. If a complete chapter is an extension, only the page heading is shaded. These chapters are:

In this publication, the term standard COBOL means American National Standard COBOL; the term IBM Full American National Standard COBOL means this IBM implementation of American National Standard COBOL and all extensions to that standard. There are two types of extensions:

- DOS/VS COBOL Considerations
- Subprogram Linkage Statements
- Debugging Language
- Format Control of the Source Program Listing
- Sterling Currency

1. Those that represent features not specified by American National Standard COBOL.
2. Those that represent an easing of the strict American National Standard

Seventh Edition (April 1976)

This edition is a reprint of GC28-6394-5 incorporating changes released in Technical Newsletter GN26-0801 (dated November 1, 1975).

This edition, as amended by Technical Newsletter GN26-0887, describes Version 2 of IBM DOS Full American National Standard COBOL at the Release 26 level of the Disk Operating System. It also describes the Program Product Version 3, Release 3, including amended System/370 device support, and Release 2 of the Program Product DOS/VS COBOL.

Information in this publication is subject to significant change. Therefore, before using this publication, consult the latest *IBM System/360 Bibliography*, GC20-0360, and *IBM System/370 Bibliography*, GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are current and applicable.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150. Comments and suggestions become the property of IBM.

For the less experienced programmer, the introduction summarizes the general principles of COBOL, highlights features of American National Standard COBOL, and, through an example, illustrates the logical sequence and interrelationship of commonly used elements of a COBOL program. The balance of the publication gives the specific rules for correct programming in IBM Full American National Standard COBOL, as implemented by the System/360 Disk Operating System. Appendixes provide supplemental information useful in writing COBOL programs. Appendix A describes the use of intermediate results in arithmetic operations; Appendix B contains several sample programs showing the use of mass storage files; Appendix C lists all of the formats and reserved words in IBM Full American National Standard COBOL; Appendix D is a file processing summary; Appendix E gives considerations for the use of ASCII encoded files; Appendix F explains the symbolic debugging feature; Appendix G explains combined function card processing.

Compiler output and restrictions, programming examples, and information on running an IBM American National Standard COBOL program are found in the publication IBM DOS Full American National Standard COBOL Programmer's Guide, Order No. GC28-6398 and in the Program Product publications:

IBM DOS Full American National Standard COBOL Compiler and Library, Version 3, Programmer's Guide, Order No. SC28-6441

IBM DOS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6478

These programmer's guides and this language reference manual are corequisite publications.

A knowledge of basic data processing techniques is mandatory for the

understanding of this publication. Such information, as it applies to System/360, can be found in the following publications:

Introduction to IBM Data Processing Systems, Order No. GC20-1684

Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods, Order No. GC20-1649

The reader should also have a general knowledge of COBOL before using this manual. Useful background information can be found in the following publications:

American National Standard COBOL Coding:

Card And Tape Applications Text, Order No. SR29-0283

Coding Techniques And Disk Applications Text, Order No. SR29-0284

Illustrations, Order No. SR29-0285

Student Reference Guide, Order No. SR29-0286

Where information in the foregoing publications conflicts with information in this publication, the contents herein supersede any other in the writing of COBOL programs. Any violation of the rules defined in this publication for using the Disk Operating System is considered an error.

A general knowledge of the IBM Disk Operating System is desirable, although not mandatory. The following publication gives such information:

IBM System/360 Disk and Tape Operating System: Concepts and Facilities, Order No. GC24-5030.

ACKNOWLEDGMENT

The following extract from Government Printing Office Form Number 1965-0795689 is presented for the information and guidance of the user:

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgment of the source, but need not quote this entire section.

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

"No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (Trademark of Sperry Rand Corporation),
Programming for the UNIVAC (R) I and II, Data Automation
Systems copyrighted 1958, 1959, by Sperry Rand
Corporation; IBM Commercial Translator, Form
No. F28-8013, copyrighted 1959 by IBM; FACT, DSI
27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications."

Date of Publication: December 3, 1976

Form of Publication: TNL GN26-0887 to GC28-6394-4, -5, -6

IBM DOS COBOL

Maintenance: Documentation

- Minor technical changes and additions have been made to the text.

Date of Publication: March 15, 1974

Form of Publication: TNL GN28-1062 to GC28-6394-4

IBM DOS/VS COBOL

New: Programming Features

- SORT-OPTION clause for Sort and Merge Features
- 5425 MFCU Support

Maintenance: Documentation only

Minor technical changes and corrections to update the documentation to Release 2

IBM DOS Full American National Standard COBOL, Versions 2 and 3

Maintenance: Documentation only

- 5425 MFCU support deleted
- Minor technical changes and corrections

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

Date of Publication: October 15, 1973

Form of Publication: TNL GN28-1047 to GC28-6394-4

IBM DOS/VS COBOL

New: Programming Features

- Merge Facility

New: Documentation only

- Miscellaneous File Processing Considerations

Maintenance: Documentation only

Minor technical changes to update the documentation to the initial release level.

IBM DOS Full American National Standard COBOL, Versions 2 and 3

Maintenance: Documentation only

Minor technical changes and corrections.

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

Date of Publication: March 15, 1974

Form of Publication: TNL GN28-1062 to GC28-6394-4

IBM DOS/VS COBOL

New: Programming Features

- SORT-OPTION clause for Sort and Merge Features
- 5425 MFCU Support

Maintenance: Documentation only

Minor technical changes and corrections to update the documentation to Release 2

IBM DOS Full American National Standard COBOL, Versions 2 and 3

Maintenance: Documentation only

- 5425 MFCU support deleted
- Minor technical changes and corrections

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

Date of Publication: October 15, 1973

Form of Publication: TNL GN28-1047 to GC28-6394-4

IBM DOS/VS COBOL

New: Programming Features

- Merge Facility

New: Documentation only

- Miscellaneous File Processing Considerations

Maintenance: Documentation only

Minor technical changes to update the documentation to the initial release level.

IBM DOS Full American National Standard COBOL, Versions 2 and 3

Maintenance: Documentation only

Minor technical changes and corrections.

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

Date of Publication: May 15, 1972

Form of Publication: TNL GN28-0489 to GC28-6394-2

IBM DOS Full American National Standard COBOL, Version 3

New: Programming Features

- Added System/370 device support: 2319, 3211, 3330, 3410, 3420, 3505, 3525

New: Documentation Only

- Symbolic Debug example
- 3525 Combined Function Processing

Miscellaneous Changes for Versions 2 and 3

Maintenance: Documentation only

Minor technical changes and corrections

Date of Publication: April 15, 1971

Form of Publication: TNL GN28-0436 to GC28-6394-2

IBM DOS Full American National Standard COBOL, Version 3

New: Programming Features

- ASCII Tape file processing
- SIGN clause implementation
- OBJECT-COMPUTER paragraph requests System/370 instructions
- ON statement enhancement

Miscellaneous Changes for Versions 2 and 3

Maintenance: Documentation only

- PICTURE clause description and table of precedence
- USAGE clause description
- Minor technical changes and corrections
- Added device support

Summary of Amendments**Number 2**

Date of Publication: January 1970

Form of Publication: Revision, GC28-6394-2

IBM DOS Full American National Standard COBOL, Version 2

Maintenance: Documentation only

Minor technical changes and corrections

Summary of Amendments**Number 1**

Date of Publication: July 1969

Form of Publication: Revision, GC28-6394-1

Miscellaneous Changes

New: Documentation only

- Table Handling clarifications
- Table Handling sample program

Maintenance: Documentation only

Minor technical changes and corrections

DOS/VS COBOL CONSIDERATIONS	i
WHEN-COMPILED Special Register	i
Configuration Section	ii
SOURCE-COMPUTER Paragraph	ii
OBJECT-COMPUTER Paragraph	ii
SPECIAL-NAMES Paragraph	iv
VSAM File Processing	iv
Environment Division -- File-Control Paragraph	v
SELECT Clause	vi
ASSIGN Clause	vi
RESERVE Clause	vi
ORGANIZATION Clause	vii
ACCESS MODE Clause	vii
RECORD KEY Clause (Format 2)	vii
PASSWORD Clause	viii
FILE STATUS Clause	viii
Environment Division -- I-O-CONTROL Paragraph	viii
RERUN Clause	ix
SAME Clause	ix
Data Division -- FD Entry	ix
LABEL RECORDS Clause	ix
Procedure Division	ix
Common Processing Facilities	x
EXCEPTION/ERROR Declarative	xi
OPEN Statement	xiii
START Statement	xv
READ Statement	xvi
WRITE Statement	xvii
REWRITE Statement	xix
DELETE Statement	xx
CLOSE Statement	xxi
Merge Facility	xxi
Environment Division	xxii
File-Control Entry for Merge Files	xxii
I-O-Control Paragraph	xxii
Data Division	xxiii
Merge-File Description Entry	xxiii
Procedure Division	xxiv
MERGE Statement	xxiv
3886 OCR Processing	xxvii
FIPS Flagger	xxviii
Miscellaneous File Processing Considerations	xxxiv
File Processing Summary -- New Devices	xxxiv
ASSIGN Clause	xxxiv
ACTUAL KEY Clause	xxxv
APPLY WRITE-VERIFY Clause	xxxv
APPLY CYL-OVERFLOW Clause	xxxv
BLOCK CONTAINS Clause	xxxv
LABEL RECORDS Clause	xxxv
ERROR Declarative	xxxv
ACCEPT Literal Statement	xxxvi
OPEN Statement	xxxvi
WRITE Statement	xxxvi
CLOSE Statement	xxxvi
Sort Feature	xxxvi
5425 Combined Function Processing	xxxvi

FIGURES

Figure I. Moves and Comparisons -- System/360 vs. System/370 . . .	iii
Figure II. Shift And Round Decimal (SRP) -- System/360 vs. System/370	iv
Figure III. Status Key Values and Their Meanings	x
Figure IV. OPEN Statement Options and Permissible I/O Statements . . .	xiv
Figure V. KEY Item Categories and Collating Sequences	xxvi
Figure VI. The Four Levels of FIPS Processing	xxix
Figure VII. File Processing Summary -- DOS/VS COBOL Devices . . .	xxxiv
Figure VIII. Error Bytes for 3540 -- GIVING Option	xxxvi

CONTENTS -- AMERICAN NATIONAL STANDARD COBOL

FEATURES OF THE DOS FULL COBOL PROGRAM	
PRODUCT COMPILERS	11
INTRODUCTION	15
Principles of COBOL	16
A Sample COBOL Program	18
Identification Division	19
Environment Division	19
Data Division	20
Procedure Division	23
Beginning the Program -- Input Operations	23
Arithmetic Statements	24
Conditional Statements	25
Handling Possible Errors	26
Data-Manipulation Statements	26
Output Operations	27
Procedure Branching Statements	28
Ending the Program	31
PART I -- LANGUAGE CONSIDERATIONS	35
STRUCTURE OF THE LANGUAGE	37
COBOL Character Set	37
Characters Used in Words	37
Characters Used for Punctuation	38
Characters Used for Editing	39
Characters Used in Arithmetic Expressions	39
Characters Used for Relation-conditions	39
Types of Words	39
Reserved Words	40
Names	41
Special-Names	41
Constants	41
Literals	41
Figurative Constants	43
Special Registers	44
ORGANIZATION OF THE COBOL PROGRAM	46
Structure of the COBOL Program	46
METHODS OF DATA REFERENCE	48
Qualification	48
Subscripting	49
Indexing	49
USE OF THE COBOL CODING FORM	50
Sequence Numbers	50
Area A and Area B	50
Division Header	51
Section Header	51
Paragraph-names and Paragraphs	51
Level Indicators and Level Numbers	51
Continuation of Lines	51
Continuation of Nonnumeric Literals	52
Continuation of Words and Numeric Literals	52
Blank Lines	52
Comment Lines	52
FORMAT NOTATION	53
PART II -- IDENTIFICATION AND ENVIRONMENT DIVISIONS	55
IDENTIFICATION DIVISION	57
PROGRAM-ID Paragraph	57
DATE-COMPILED Paragraph	58
ENVIRONMENT DIVISION -- FILE PROCESSING SUMMARY	59
Data Organization	59
Sequential Data Organization	59
Direct Data Organization	59
Indexed Data Organization	60
Access Methods	60
Accessing a Sequential File	60
Accessing a Direct File	60
Sequential Access	60
Random Access	61
Accessing an Indexed File	61
Sequential Access	61
Random Access	61
ORGANIZATION OF THE ENVIRONMENT DIVISION	63
ENVIRONMENT DIVISION -- CONFIGURATION SECTION	64
SOURCE-COMPUTER Paragraph	64
OBJECT-COMPUTER Paragraph	65
Program Product Information -- Version 3	65
SPECIAL-NAMES Paragraph	65
ENVIRONMENT DIVISION -- INPUT-OUTPUT SECTION	68
FILE-CONTROL Paragraph	68
SELECT Clause	69
ASSIGN Clause	69
Program Product Information -- Version 3	72
RCE and OMR Format Descriptor	75
RESERVE Clause	77
Program Product Information -- Version 3	77
FILE-LIMIT Clause	77
ACCESS MODE Clause	78
PROCESSING MODE Clause	78
ACTUAL KEY Clause	79
NOMINAL KEY Clause	82
RECORD KEY Clause	83
TRACK-AREA Clause	83
I-O-CONTROL Paragraph	84
RERUN Clause	84
SAME Clause	85
MULTIPLE FILE TAPE Clause	86
APPLY Clause	87
PART III -- DATA DIVISION	91

DATA DIVISION -- INTRODUCTION	93	Imperative Statements151
Organization of External Data	93	Compiler-Directing Statements152
Description of External Data	93	ARITHMETIC EXPRESSIONS153
ORGANIZATION OF THE DATA DIVISION	94	Arithmetic Operators153
Organization of Data Division Entries	95	CONDITIONS155
Level Indicator	95	Test Conditions155
Level Number	95	Class Condition156
Special Level Numbers	96	Condition-Name Condition157
Indentation	97	Relation Condition158
File Section	97	Sign Condition162
File Description Entry	97	Switch-Status Condition162
Record Description Entry	97	Compound Conditions162
Working-Storage Section	98	Evaluation Rules163
Data Item Description Entries	98	Implied Subjects and	
Record Description Entries	98	Relational-Operators164
Linkage Section	98	Implied Subject165
Report Section	99	Implied Subject and Relational	
FILE DESCRIPTION ENTRY -- DETAILS OF		Operator165
CLAUSES100	Implied Subject, and Subject and	
BLOCK CONTAINS Clause100	Relational-Operator165
RECORD CONTAINS Clause102	CONDITIONAL STATEMENTS166
Recording Mode103	IF Statement166
RECORDING MODE Clause104	Nested IF Statements167
LABEL RECORDS Clause105	DECLARATIVES169
VALUE OF Clause106	Sample Label Declarative Program172
DATA RECORDS Clause106	ARITHMETIC STATEMENTS179
REPORT Clause107	CORRESPONDING Option179
DATA DESCRIPTION108	GIVING Option179
DATA DESCRIPTION ENTRY -- DETAILS OF		ROUNDED Option179
CLAUSES111	SIZE ERROR Option180
Data-name or FILLER Clause111	Overlapping Operands180
REDEFINES Clause112	ADD Statement181
BLANK WHEN ZERO Clause115	COMPUTE Statement182
JUSTIFIED Clause116	DIVIDE Statement183
OCCURS Clause116	MULTIPLY Statement184
PICTURE Clause116	SUBTRACT Statement185
The Three Classes of Data117	PROCEDURE BRANCHING STATEMENTS187
Character String and Item Size118	GO TO Statement187
Repetition of Symbols118	ALTER Statement188
Symbols Used in the PICTURE Clause118	PERFORM Statement189
The Five Categories of Data120	STOP Statement196
Types of Editing125	EXIT Statement196
Insertion Editing125	DATA-MANIPULATION STATEMENTS198
Zero Suppression and Replacement		MOVE Statement198
Editing128	EXAMINE Statement201
Program Product Information --		TRANSFORM Statement203
Version 3129	INPUT/OUTPUT STATEMENTS206
SIGN Clause129	OPEN Statement206
SYNCHRONIZED Clause130	START Statement208
Slack Bytes132	SEEK Statement210
USAGE Clause136	READ Statement211
Display Option137	WRITE Statement212
The Computational Options138	Program Product Information	
Program Product Information --		(Version 3)216
Version 3139	REWRITE Statement218
VALUE Clause142	ACCEPT Statement219
RENAMES Clause144	DISPLAY Statement220
PART IV -- PROCEDURE DIVISION147	CLOSE Statement221
ORGANIZATION OF THE PROCEDURE DIVISION149	Sequential File Processing222
Categories of Statements150	Random File Processing224
Conditional Statements151		

SUBPROGRAM LINKAGE STATEMENTS226	Procedure Division Considerations273
CALL Statement226	GENERATE Statement273
ENTRY Statement227	Detail Reporting273
USING Option228	Summary Reporting273
Program Termination Considerations231	Operation of the GENERATE Statement274
EXIT PROGRAM Statement232	INITIATE Statement275
GOBACK Statement232	TERMINATE Statement275
STOP RUN Statement232	USE Sentence276
COMPILER-DIRECTING STATEMENTS233	Special Registers: PAGE-COUNTER and	
COPY Statement233	LINE-COUNTER277
ENTER Statement233	PAGE-COUNTER277
NOTE Statement233	LINE-COUNTER278
PART V -- SPECIAL FEATURES235	Sample Report Writer Program279
SORT FEATURE237	Key Relating Report to Report	
Elements of the Sort Feature237	Writer Source Program282
Environment Division Considerations		TABLE HANDLING FEATURE289
for Sort238	Subscripting289
Input-Output Section238	Indexing290
File-Control Paragraph238	Restrictions on Indexing,	
Assignment of Sort Work Units239	Subscripting, and Qualification291
I-O-CONTROL Paragraph239	Example of Subscripting and Indexing291
RERUN Clause240	Data Division Considerations for Table	
SAME RECORD/SORT AREA Clause240	Handling292
Data Division Considerations for Sort241	OCCURS Clause292
File Section241	USAGE IS INDEX Clause299
Sort-File Description241	Procedure Division Considerations for	
Procedure Division Considerations for		Table Handling300
Sort242	Relation Conditions300
SORT Statement242	SEARCH Statement301
RELEASE Statement247	SET Statement306
RETURN Statement248	Sample Table Handling Program307
EXIT Statement248	SEGMENTATION FEATURE309
Special Registers for Sort249	Organization309
Sample Program Using the Sort Feature250	Fixed Portion309
REPORT WRITER FEATURE252	Independent Segments309
Data Division -- Overall Description252	Segment Classification310
Procedure Division -- Overall		Segmentation Control310
Description253	Structure of Program Segments310
Data Division Considerations for		Priority Numbers310
Report Writer254	Segment Limit311
File Description254	Restrictions on Program Flow312
REPORT Clause254	ALTER Statement312
RECORDING MODE Clause255	PERFORM Statement312
DATA RECORDS Clause255	Called Programs312
RECORD CONTAINS Clause255	SOURCE PROGRAM LIBRARY FACILITY313
Report Section256	COPY Statement313
Report Description Entry256	Extended Source Program Library	
CODE Clause256	Facility316
CONTROL Clause257	BASIS Card316
PAGE LIMIT Clause258	INSERT Card316
Report Group Description Entry261	DELETE Card316
LINE Clause263	DEBUGGING LANGUAGE318
NEXT GROUP Clause265	READY/RESET TRACE Statement318
TYPE Clause267	EXHIBIT Statement318
USAGE Clause269	ON (Count-conditional) Statement320
COLUMN Clause269	Program Product Information --	
GROUP INDICATE Clause270	Version 3321
JUSTIFIED Clause270	Compile-Time Debugging Packet322
PICTURE Clause270	DEBUG Card322
RESET Clause270	FORMAT CONTROL OF THE SOURCE PROGRAM	
BLANK WHEN ZERO Clause271	LISTING323
SOURCE, SUM, or VALUE Clause271	EJECT Statement323
		SKIP1, SKIP2, and SKIP3 Statements323

STERLING CURRENCY FEATURE AND INTERNATIONAL CONSIDERATIONS	324	Data Description Entries	357
Sterling Nonreport	325	PICTURE Clause	357
Sterling Sign Representation	326	SIGN Clause	357
Sterling Report	327	USAGE Clause	357
Procedure Division Considerations	330	III -- Procedure Division	357
International Considerations	330	LABEL PROCEDURE Declarative	358
SUPPLEMENTARY MATERIAL	331	Relation Conditions	358
APPENDIX A: INTERMEDIATE RESULTS	333	IV -- Sort Feature	360
Compiler Calculation of Intermediate Results	333	Environment Division	360
APPENDIX B: SAMPLE PROGRAMS	335	ASSIGN Clause	360
Creation of a Direct File	336	RERUN Clause	361
Creation of an Indexed File	338	Data Division	361
Random Retrieval and Updating of an Indexed File	339	SIGN Clause	361
APPENDIX C: AMERICAN NATIONAL STANDARD COBOL FORMAT SUMMARY AND RESERVED WORDS	341	USAGE Clause	361
APPENDIX D: SUMMARY OF FILE-PROCESSING TECHNIQUES AND APPLICABLE STATEMENTS AND CLAUSES	351	APPENDIX F: SYMBOLIC DEBUGGING FEATURE	363
APPENDIX E: ASCII CONSIDERATIONS	355	Object-Time Control Cards	363
I -- Environment Division	355	Sample Program -- TESTRUN	365
ASSIGN Clause	355	Debugging TESTRUN	366
RERUN Clause	356	APPENDIX G: COMBINED FUNCTION CARD PROCESSING	379
II -- Data Division	356	I -- Environment Division Considerations	379
File Section	356	SPECIAL-NAMES Paragraph	379
BLOCK CONTAINS Clause	356	SELECT Clause	380
LABEL RECORDS Clause	356	ASSIGN Clause	380
RECORDING MODE Clause	357	RESERVE Clause	381
Compiler Calculation of Recording Mode	357	II -- Data Division Considerations	381
		III -- Procedure Division Considerations	382
		OPEN Statement	382
		READ Statement	382
		WRITE Statement -- Punch Function Files	382
		WRITE Statement -- Print Function Files	383
		CLOSE Statement	384
		IBM AMERICAN NATIONAL STANDARD COBOL GLOSSARY	385
		INDEX	399

FIGURES

Figure 1. Typical Ledger Records Used for MASTER-RECORD	21	Figure 32. Logical Flow of Option 4 PERFORM Statement Varying Three Identifiers195
Figure 2. Typical DETAIL-RECORD	22	Figure 33. Permissible Moves200
Figure 3. Illustration of Procedure Branching	29	Figure 34. Examples of Data Examination202
Figure 4. Complete UPDATING Program (Part 1 of 2)	33	Figure 35. Examples of Data Transformation203
Figure 5. Reference Format	50	Figure 36. Combinations of FROM and TO Options (Part 1 of 2)204
Figure 6. Summary of File-Processing Techniques	62	Figure 37. Action Taken for Function-Names -- ADVANCING Option . .	.214
Figure 7. Choices of Function-name-1 and Action Taken	66	Figure 38. Values of Identifier-2 and Their Interpretation -- POSITIONING Option215
Figure 8. Values of Organization Field for File Organization	72	Figure 39. Values of Integer and Their Interpretations -- POSITIONING Option .	.215
Figure 10. Structure of the First Eight Bytes of ACTUAL KEY -- Actual Track Addressing	80	Figure 40. Relationship of Types of Sequential Files and the Options of the CLOSE Statement224
Figure 11. Level Indicator Summary	95	Figure 41. Relationship of Types of Random Files and the Options of the CLOSE Statement225
Figure 12. Areas REDEFINED without Changes in Length113	Figure 42. Effect of Program Termination Statements Within Main Programs and Subprograms231
Figure 13. Areas REDEFINED and Rearranged114	Figure 43. SORT Collating Sequences Used for Sort Keys244
Figure 14. Class and Category of Elementary and Group Data Items117	Figure 44. Sample Program Using the SORT Feature (Part 1 of 2)250
Figure 15. Precedence of Symbols Used in the PICTURE Clause121	Figure 45. Page Format When the PAGE LIMIT Clause is Specified260
Figure 16. Editing Sign Control Symbols and their Results126	Figure 46. Sample Program Using the Report Writer Feature (Part 1 of 4) . .	.279
Figure 17. Insertion of the Intra-occurrence Slack Bytes133	Figure 47. Report Produced by Report Writer Feature (Part 1 of 5)284
Figure 18. Insertion of Inter-occurrence Slack Bytes134	Figure 48. Storage Layout for PARTY-TABLE292
Figure 19. Internal Representation of Numeric Items (Part 1 of 2)140	Figure 49. Index-names and Index Data Items -- Permissible Comparisons301
Figure 20. Permissible Symbol Pairs -- Arithmetic Expressions154	Figure 50. Format 1 SEARCH Operation Containing Two WHEN Options304
Figure 21. Valid Forms of the Class Test156	Figure 51. Sample Table Handling Program (Part 1 of 2)307
Figure 22. Relational-operators and Their Meanings158	Figure 52. Sterling Currency Editing Applications329
Figure 23. Permissible Comparisons161	Figure 53. Compiler Action on Intermediate Results334
Figure 24. Logical Operators and the Resulting Values upon Evaluation163	Figure 54. Using the TRANSFORM Statement with ASCII Comparisons359
Figure 25. Permissible Symbol Pairs -- Compound Conditions164	Figure 55. EBCDIC and ASCII Collating Sequences for COBOL Characters -- in Ascending Order360
Figure 26. Conditional Statements with Nested IF Statements167	Figure 56. Individual Type Codes Used in SYMDMP Output367
Figure 27. Logical Flow of Conditional Statement with Nested IF Statements . .	.168	Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 1 of 11)367
Figure 28. Error Byte Meaning for the GIVING Option of an Error Declarative .	.176	Figure 58. Identifier-2 Stacker Values for WRITE AFTER POSITIONING383
Figure 29. File Processing Techniques and Associated Error Declaratives Capabilities178		
Figure 30. Logical Flow of Option 4 PERFORM Statement Varying One Identifier193		
Figure 31. Logical Flow of Option 4 PERFORM Statement Varying Two Identifiers194		

Special DOS/VS COBOL considerations are discussed in the following pages. Implementation areas described are:

- WHEN-COMPILED Special Register
- The Configuration Section
- VSAM (Virtual Storage Access Method) Processing
- Merge Facility -- with SORT-OPTION clause
- 3886 OCR (Optical Character Reader) Processing
- FIPS (Federal Information Processing Standard) Flagger
- Miscellaneous File Processing Considerations

DOS/VS COBOL supports all of the additional features described in this chapter. Support for these features is provided through a subset of the complete COBCL language as documented in CODASYL COBOL Journal Of Development. IBM-specified language capabilities are also implemented. All features of DOS Full American National Standard COBOL, Version 3, continue to be supported.

Compiler output and restrictions, programming examples, and information on running an IBM DOS/VS COBOL program are found in the following Program Product publication:

IBM DOS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6478

Additional information on DOS/VS can be found in the following publications:

Introduction to DOS/VS, Order No. GC33-5370

DOS/VS Systems Management Guide, Order No. GC33-5371

DOS/VS Data Management Guide, Order No. GC33-5372

DOS/VS Access Method Services, Order No. GC33-5832

WHEN-COMPILED SPECIAL REGISTER

The WHEN-COMPILED special register is provided as a maintainability aid for the user; it makes available to the object program the date-and-time-compiled constant carried in the object module.

WHEN-COMPILED is a 16-byte alphanumeric field valid only as the sending field in a MOVE statement. The format of these sixteen bytes is MM/DD/YYhh.mm.ss (MONTH/DAY/YEARhour.minute.second) or DD/MM/YYhh.mm.ss (DAY/MONTH/YEARhour.minute.second).

This special register is a programmer aid that provides a means of associating a compilation listing with both the object program and the output produced at execution time.

SOURCE/OBJECT-COMPUTER-Paragraphs (DOS/VS)

CONFIGURATION SECTION

The Configuration Section describes the computer on which the source program is compiled, the computer on which the object program is executed, and, optionally, SPECIAL-NAMES, which relate function-names used by the compiler with user-specified mnemonic-names.

```

                                     General Format
-----
CONFIGURATION SECTION.

SOURCE-COMPUTER.  computer-name.

OBJECT-COMPUTER.  computer-name

      [MEMORY SIZE integer { WORDS
                             CHARACTERS } ]
                             MODULES

      [SEGMENT-LIMIT IS priority-number].

SPECIAL-NAMES.  [function-name-1 IS mnemonic-name] ...
      [function-name-2 [IS mnemonic-name]
      { ON STATUS IS condition-name-1
      { OFF STATUS IS condition-name-2
      [OFF STATUS IS condition-name-2] }
      [ON STATUS IS condition-name-1] } ] ...

      [CURRENCY SIGN IS literal] [DECIMAL-POINT IS COMMA].

```

The Configuration Section and its associated paragraphs are optional in a COBOL source program.

SOURCE-COMPUTER PARAGRAPH

The SOURCE-COMPUTER paragraph describes the computer upon which the source program is to be compiled. This paragraph is treated as documentation.

Computer-name is a word in the form IBM-370[-model-number].

OBJECT-COMPUTER PARAGRAPH

The OBJECT-COMPUTER paragraph describes the computer upon which the object program is to be executed.

Computer-name must be the first entry in the OBJECT-COMPUTER paragraph. Computer-name is a word in the form IBM-370[-model-number].

System/370 instructions are provided automatically by DOS/VS COBOL. (When IBM-360 is specified, the compiler generates System/370 instructions, and issues a warning message.) The Compiler generates instructions from the System/370 set, including Move Long (MVCL), Compare Logical Long (CLCL), and Shift And Round Decimal (SRP) that are particularly useful to COBOL. These System/370 instructions replace object-time subroutines and instructions that former COBOL Compilers generated under System/360 including routines and instructions to handle decimal arithmetic scaling (where operands have a different number of decimal places) and rounding. System/370 support also gives much improved processing of variable length fields.

Since System/370 does not require boundary alignment for COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items, no moves are generated for items that are not SYNCHRONIZED.

Performance Considerations:: Space occupied by a DOS/VS COBOL program is decreased, particularly when calls to object-time subroutines, are no longer necessary. Such calls are always generated in System/360 for variable-length moves and comparisons. If there is at least one variable-length alphanumeric move in the source program, System/370 support reduces the size of the object program by at least 484 bytes; if there is at least one variable-length alphanumeric comparison, System/370 support reduces the size of the object program by at least an additional 498 bytes.

Number of Bytes in Each Move or Comparison	For Each Alphanumeric Move: Object-program Instructions		For Each Comparison (in a conditional expression): Object-program Instructions	
	System/360 Bytes Needed	System/370 Bytes Needed	System/360 Bytes Needed	System/370 Bytes Needed
Variable length	26+480*	14-22	26+496*	16-24
fixed length				
1-256	6-16	6-16	8-26	8-26
257-512	12-22	12-22	16-36	16-24
513-768	18-28	14-22	24-46	16-24
769-1024	24-34	14-22	32-56	16-24
1025-1280	30-40	14-22	40-66	16-24
1281-1536	36-46	14-22	48-76	16-24
...
>4096	26+480*	14-22	26+496*	16-24

*Bytes needed to invoke object-time subroutine, plus size of subroutine itself.

Figure I. Moves and Comparisons -- System/360 vs. System/370

Figure I gives comparative figures without right justification for fixed-length and variable-length MOVE statements, and for fixed-length and variable-length comparisons.

Figure II gives comparative figures for Shift And Round Decimal generation; the savings shown are made for each such operation in the object program.

The MEMORY SIZE clause can be used to document the actual equipment configuration needed to run the object program.

The SEGMENT-LIMIT clause is discussed in the Segmentation Chapter.

SPECIAL-NAMES Paragraph/VSAM (DOS/VS)

Except for the computer-name entry and the SEGMENT-LIMIT clause, the OBJECT-COMPUTER paragraph is treated as documentation.

Function	System/360 Bytes Needed	System/370 Bytes Needed
Rounding	39 + literal*	6
Left Scaling	6 + literal*	6
Right Scaling	12	6

*As used for decimal point alignment the literal varies in length with size of data-item, number of decimal positions defined, and/or scaling positions defined.

Figure II. Shift And Round Decimal (SRP) -- System/360 vs. System/370

SPECIAL-NAMES PARAGRAPH

The SPECIAL-NAMES paragraph as discussed in the Environment Division chapter applies to DOS/VS COBOL without change.

VSAM FILE PROCESSING

VSAM (Virtual Storage Access Method) is a high-performance access method of DOS/VS for use with direct access storage. VSAM provides high-speed retrieval and storage of data, flexible data organization, ease of conversion from other access methods, and ease of use -- including simplified job control statements, data protection against unauthorized access, central control of data management functions, device independence (freedom from consideration of block sizes, control information, record deblocking, etc.), and cross-system compatibility.

Access Method Services, a multi-function utility program is used to define a VSAM data set, and optionally load records into it, convert an existing indexed or sequential data set to VSAM format, and perform other tasks as well. Access Method Services is described in DOS/VS Access Method Services, Order No. GC33-5832.

VSAM allows key-sequenced and entry-sequenced data sets; records can be fixed or variable in length.

In a key-sequenced data set (KSDS), records are stored in the ascending collating sequence of some embedded key field. For indexed files of this type, records can be retrieved sequentially in key sequence; they can also be retrieved randomly according to the particular value of the key.

In an entry-sequenced data set (ESDS), the records are stored in the order in which they are presented for inclusion in the data set. New records are stored at the end of the data set. In COBOL, record retrieval for sequential files of this type must be sequential.

VSAM files may be written on the following mass storage devices: 2314, 2319, 3330, 3340.

For VSAM file processing in COBOL, there are special language considerations in the Environment, Data, and Procedure Division.

ENVIRONMENT DIVISION -- FILE-CONTROL PARAGRAPH

The File-Control paragraph names the VSAM file, associates it with an external medium, and allows specification of other file-related information.

General Format 1 -- Sequential VSAM Files

FILE-CONTROL.

```

{SELECT [OPTIONAL] file-name
  ASSIGN TO system-name-1 [system-name-2] ...

[RESERVE integer [ AREA
                  AREAS ] ]

[ORGANIZATION IS SEQUENTIAL]

[ACCESS MODE IS SEQUENTIAL]

[PASSWORD IS data-name-1]

[FILE STATUS IS data-name-2].} ...

```

General Format 2 -- Indexed VSAM Files

FILE-CONTROL.

```

{SELECT file-name
  ASSIGN TO system-name-1 [system-name-2] ...

[RESERVE integer [ AREA
                  AREAS ] ]

ORGANIZATION IS INDEXED

[ACCESS MODE IS { SEQUENTIAL
                  RANDOM
                  DYNAMIC } ]

RECORD KEY IS data-name-3

[PASSWORD IS data-name-1]

[FILE STATUS IS data-name-2].} ...

```

Each file described by an FD entry or SD entry in the Data Division must be described in one and only one File-Control entry.

The key word FILE-CONTROL may appear only once, at the beginning of the File-Control paragraph. The word FILE-CONTROL must begin in Area A, and be followed by a period followed by a space.

SELECT/ASSIGN Clauses (DC/VS)

Each File-Control entry must begin with a SELECT clause followed immediately by an ASSIGN clause. The order in which the other clauses appear is not significant, except that for indexed VSAM files the PASSWORD clause, if specified, must immediately follow the RECORD KEY clause. Each File-Control entry must end with a period followed by a space.

Each data-name in the File-Control entry may be qualified; it may not be subscripted or indexed. Each data-name must be at a fixed displacement from the beginning of the data description entry in which it appears; that is, it must not appear in the entry after an OCCURS DEPENDING ON clause.

SELECT Clause

The SELECT clause is used to name each file in the program. Each file described with an FD entry or SD entry in the Data Division must be named once and only once as a file-name following the key word SELECT.

FORMAT 1: The OPTIONAL clause must be specified for input files that are not necessarily present each time the object program is executed.

If file-name represents a sort file, only the ASSIGN clause may be written following the SELECT clause.

ASSIGN Clause

The ASSIGN clause associates the file with an external storage medium.

System-name specifies a system logical unit, and, optionally, a device class, a device number, the file organization, and the external name. System-name has the following structure:

SYSnnn[-class][-device][-organization][-name]

The SYSnnn field is required, and nnn must be a three-digit number from 000 through 240 inclusive. This number represents the symbolic unit to which the file is assigned.

The class and device fields are included for compatibility only; for VSAM files, these fields are treated as documentation.

The organization field is required for sequential VSAM files. The entry must be AS.

The organization field must not be specified for indexed VSAM files.

The name field is an optional three-character through seven-character field, specifying the external name by which the file is known to the system. If name is not specified, the symbolic unit (SYSnnn) is used as the external name. The name field must be specified if more than one file is assigned to the same symbolic unit.

RESERVE Clause

The RESERVE clause is treated as documentation.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of the file. The file organization is established at the time the file is defined and cannot subsequently be changed.

FORMAT 1: If the ORGANIZATION clause is omitted, ORGANIZATION SEQUENTIAL is assumed.

When ORGANIZATION SEQUENTIAL is specified or assumed, the records in the file are positioned sequentially in the order they were created. Once established, the position of the file records does not change.

FORMAT 2: When ORGANIZATION INDEXED is specified, each logical record in the file contains an embedded RECORD KEY which is associated with an index, and each record is identified through its RECORD KEY value. After records have been updated, or have been added to or deleted from the file, the position of the records may have changed.

ACCESS MODE Clause

The ACCESS MODE clause specifies the manner in which records in the file are to be processed.

When the ACCESS MODE clause is omitted, ACCESS MODE SEQUENTIAL is assumed.

When ACCESS MODE SEQUENTIAL is specified or assumed, the records are processed sequentially. That is, the next logical record in the file is the next processed.

When ORGANIZATION IS SEQUENTIAL is specified or assumed, the records in the file are processed in the sequence established when the file was created or extended.

When ORGANIZATION IS INDEXED is specified, the records in the file are processed in the sequence of ascending record key values.

FORMAT 2: For indexed VSAM files, ACCESS MODE RANDOM and ACCESS MODE DYNAMIC can also be specified.

When ACCESS MODE RANDOM is specified, the sequence in which records are processed is determined by the sequence in which record keys are presented. The desired record is accessed by placing the value of its key in the RECORD KEY data item before the associated input/output statement is executed.

When ACCESS MODE DYNAMIC is specified, records in the file are processed either sequentially and/or randomly. The form of the specific input/output request determines the access mode.

RECORD KEY Clause (Format 2)

The RECORD KEY clause specifies the data item within the record which contains the key for that record. A RECORD KEY must be specified for an indexed VSAM file.

Data-name-3 is the RECORD KEY data item. Data-name-3 must be defined as a fixed length alphanumeric or unsigned external-decimal numeric data item within a record description entry associated with file-name. Data-name-3 is treated as an alphanumeric item.

VSAM PASSWORD/FILE STATUS Clauses/I-O-CONTROL (DOS/VS)

The value contained in data-name-3 must be unique among records in the file.

The data description of data-name-3 and its relative location in the record must be the same as that specified when the file was defined.

PASSWORD Clause

The PASSWORD clause controls object-time access to the file.

Data-name-1 is the password data item; it must be defined in the Working-Storage Section as an alphanumeric item. The first 8 characters are used as the password; a shorter field is padded with blanks to 8 characters. The password data item must be equivalent to the one externally specified.

When the PASSWORD clause is specified, at object time the password data item must contain the valid password for this VSAM file before the file can be successfully opened. (See "Status Key" in the following Common Processing Facilities description.)

FILE STATUS Clause

The FILE STATUS clause allows the user to monitor the execution of each input/output request for the file.

Data-name-2 is the Status Key data item. Data-name-2 must be defined in the Data Division as a two-character alphanumeric or unsigned external-decimal numeric item. Data-name-2 must not be defined in the File Section or the Report Section. Data-name-2 is treated as an alphanumeric item.

When the FILE STATUS clause is specified, a value is moved into the Status Key by the system after each input/output request that explicitly or implicitly refers to this file. The value indicates the status of the execution of the statement. (See "Status Key" in the following Common Processing Facilities description.)

ENVIRONMENT DIVISION -- I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph specifies the special input/output techniques to be used in the program. The I-O-CONTROL paragraph and its associated clauses are optional.

```
-----  
General Format -- VSAM Files  
-----  
I-O-CONTROL.  
  [RERUN ON system-name EVERY integer RECORDS  
    OF file-name-1] ...  
  [SAME [RECORD] AREA  
    FOR file-name-2 [file-name-3] ...] ... .
```

The key word I-O-CONTROL must begin in Area A and be followed by a period and a space.

RERUN Clause

System-name is specified as described in the Environment Division chapter; the checkpoint file must be a standard sequential file (it may not be a sequential VSAM file). The device field may not specify 3540.

File-name may specify a VSAM file.

SAME Clause

The SAME RECORD AREA clause for VSAM files is implemented as described in the Environment Division chapter.

For VSAM files, the SAME AREA clause has the same meaning as the SAME RECORD AREA clause.

DATA DIVISION -- FD ENTRY

In the FD entry for a VSAM file, the RECORD CONTAINS clause is implemented as described in the Data Division chapter.

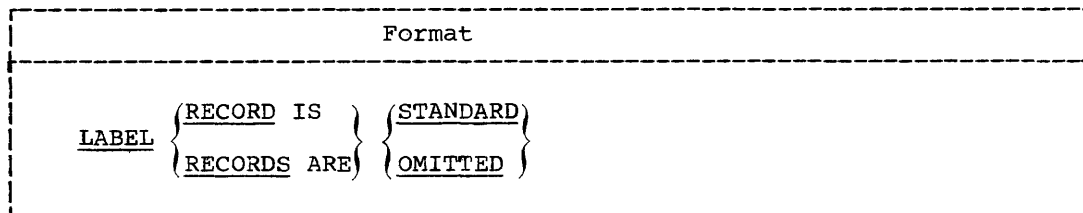
The BLOCK CONTAINS, DATA RECORDS, and VALUE OF clauses, are treated as documentation for VSAM files.

The RECORDING MODE and REPORT clauses must not be specified for VSAM files.

There are special considerations for the LABEL RECORDS clause.

LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether standard labels are present or omitted, and serves only as documentation.



For VSAM files, either the STANDARD or the OMITTED option may be specified. Either option is treated as documentation.

The LABEL RECORDS clause is required in every FD entry.

PROCEDURE DIVISION

For VSAM files, there are several Common Processing Facilities that apply to more than one input/output statement. These Common Processing Facilities are discussed before the descriptions of the separate input/output verbs.

Common Processing Facilities

★ CURRENT RECORD POINTER: Conceptually, the Current Record Pointer specifies the next record to be accessed by a sequential request. The setting of the Current Record Pointer is affected only by the OPEN, START, and READ statements. The concept of the Current Record Pointer has no meaning for random access or for output files.

★ STATUS KEY: If the FILE STATUS clause is specified in the File Control Entry, a value is placed into the specified Status Key (the 2-character data item named in the FILE STATUS clause) during execution of any request on that file; the value indicates the status of that request. The value is placed in the Status Key before execution of any Error Declarative or INVALID KEY/AT END option associated with the request.

The first character of the Status Key is known as Status Key 1; the second character is known as Status Key 2. Combinations of possible values and their meanings are shown in Figure III. See the DOS/VS Programmer's Guide for more information.

Status Key 1 Value	Meaning	Status Key 2 Value	Meaning
0	Successful Completion <i>C'00'</i>	0	No Further Information
1	At End (no next logical record, or an OPTIONAL file not available at OPEN time) <i>C'10'</i>	0	No Further Information
2	Invalid Key	1	Sequence Error
		2	Duplicate Key
		3	No Record Found
		4	Boundary Violation (indexed VSAM file)
3	Permanent Error (data check, parity check, transmission error)	0	No Further Information
		4	Boundary Violation (sequential VSAM file)
9	IBM-defined	1	Password Failure
		2	Logic Error
		3	Resource Not Available
		4	No Current Record Pointer For Sequential Request
		5	Invalid or Incomplete File Information
		6	No DLBL card
Z	User-defined	1-9 A-Z	Reserved for user purposes

Figure III. Status Key Values and Their Meanings



INVALID KEY CONDITION: The INVALID KEY condition can occur during execution of a START, READ, WRITE, REWRITE, OR DELETE statement. (For details of the causes for the condition, see documentation for those statements.) When the INVALID KEY condition is recognized, the following actions are taken in the following order:

1. If the FILE-STATUS clause is specified, a value is placed into the Status Key to indicate an INVALID KEY condition.
2. If the INVALID KEY option is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not executed.
- but*
3. If the INVALID KEY option is not specified, but an EXCEPTION/ERROR declarative procedure is specified for the file, the EXCEPTION/ERROR procedure is executed.

When an INVALID KEY condition occurs, the input/output statement which caused the condition is unsuccessful.

INTO/FROM IDENTIFIER OPTION: This option is valid for READ, REWRITE, and WRITE statements.

The INTO identifier option makes a READ statement equivalent to

```
READ file-name
MOVE record-name TO identifier
```

⊖ After successful execution of the READ statement, the current record becomes available both in the record-name and identifier.

The FROM identifier option makes a REWRITE or WRITE statement equivalent to

```
MOVE identifier TO record-name
{ REWRITE }
{ WRITE   } record-name
```

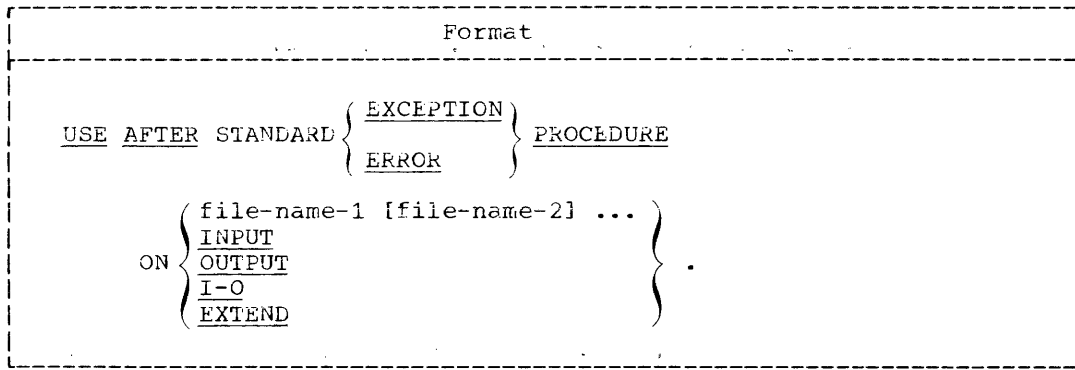
⊖ After successful execution of the WRITE or REWRITE statement, the current record may no longer be available in record-name, but is still available in identifier.

In all cases, identifier must be the name of an entry in the Working-Storage Section, the Linkage Section, or of a record description for another previously opened file. Record-name/file-name and identifier must not refer to the same storage area.

EXCEPTION/ERROR Declarative

The EXCEPTION/ERROR Declarative specifies procedures for input/output exception or error handling that are to be executed in addition to the standard system procedures.

VSAM EXCEPTION/ERROR Declarative (DOS/VS)



A USE statement, when present, must immediately follow a section header in the Declaratives Section (see "Declaratives" in the Procedure Division chapter). A USE statement must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that specify the procedures to be used.

The USE statement itself is not an executable statement; it merely defines the conditions for execution of the procedural paragraphs.

The words EXCEPTION and ERROR are synonymous and may be used interchangeably.

When the file-name option is specified, the procedure is executed only for the file(s) named. Appearance of a file-name must not cause simultaneous requests for the execution of more than one EXCEPTION/ERROR procedure. No file-name can refer to a sort file.

When the INPUT option is specified, the procedure is executed for all files opened in INPUT mode.

When the OUTPUT option is specified, the procedure is executed for all files opened in the OUTPUT mode.

When the I-O option is specified, the procedure is executed for all files opened in I-O mode.

When the EXTEND option is specified, the procedure is executed for all files opened in EXTEND mode.

The EXCEPTION/ERROR procedure is executed:

- Either after completing the standard system input/output error routine, or
- Upon recognition of an INVALID KEY or AT END condition when an INVALID KEY or AT END option has not been specified in the input/output statement, or
- Upon recognition of an IBM-defined condition which causes status key 1 to be set to 9.

After execution of the EXCEPTION/ERROR procedure, control is returned to the invoking routine.

The EXCEPTION/ERROR procedures are activated when an input/output error occurs during execution of a READ, WRITE, REWRITE, START, or DELETE statement. **OR**

If an OPEN statement is issued for a file already in the open status, the EXCEPTION/ERROR procedures are activated; when the execution of an OPEN statement is unsuccessful due to any other cause, the EXCEPTION/ERROR procedures are not activated.

If a file is in the OPEN status, and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedures are activated. If the file is in a closed status and a CLOSE statement is issued, the EXCEPTION/ERROR procedures are not activated.

Within a declarative procedure, there must be no references to nondeclarative procedures. In nondeclarative procedures, there must be no references to declarative procedures, except that PERFORM statements may refer to procedure-names associated with a declarative procedure.

OPEN Statement

The OPEN statement initiates the processing of VSAM files.

Format				
OPEN	(INPUT	file-name-1	[file-name-2] ...)
		OUTPUT	file-name-1	[file-name-2] ...)
		I-O	file-name-1	[file-name-2] ...)
		EXTEND	file-name-1	[file-name-2] ...)
				...)

At least one of the options INPUT, OUTPUT, I-O, or EXTEND must be specified; there may be not more than one instance of each option specified in one OPEN statement, although more than one file-name may be specified with each option. The INPUT, OUTPUT, I-O, and EXTEND options may appear in the any order.

Each file-name designates a file upon which the OPEN statement is to operate. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort file. The FD entry for the file must be equivalent to the information specified when the file was defined.

The successful execution of an OPEN statement determines the availability of the file and results in that file being in open mode. Before successful execution of the OPEN statement for a given file, no statement can be executed which refers explicitly or implicitly to that file. The successful execution of the OPEN statement makes the associated record area available to the program; it does not obtain or release the first data record.

The INPUT option permits opening the file for input operations.

The I-O option permits opening the file for both input and output operations.

The INPUT and I-O options are valid only for files which contain or which have contained records, whether or not the files still contain any records when the OPEN statement is executed. (That is, even if all the records in a file have been deleted, that file can still be opened INPUT or I-O.) The INPUT and I-O options must not be specified when the file has not been already created.

The OUTPUT option permits opening the file for output operations. This option can be specified when the file is being created. (The OUTPUT option must not be specified for a file which contains records, or which has contained records that have been deleted.)

The EXTEND option permits opening the file for output operations. ACCESS MODE SEQUENTIAL must be explicitly or implicitly specified. When EXTEND is specified, execution of the OPEN statement prepares the file for the addition of records immediately following the last record in the

file. Subsequent WRITE statements add records to the file, as if the file had been opened OUTPUT. The EXTEND option can be specified when a file is being created; it can also be specified for a file which contains records, or which has contained records that have been deleted.

The OPEN mode, the ACCESS MODE, and the file ORGANIZATION determine the valid input/output statements for a given VSAM file. Figure IV shows permissible combinations.

File Organization and OPEN mode		INDEXED				SEQUENTIAL			
		INPUT	OUTPUT	I-O	EXTEND	INPUT	OUTPUT	I-O	EXTEND
SEQUENTIAL	OPEN	P	P	P	P	P	P	P	P
	READ	P	-	P	-	P	-	P	-
	WRITE	-	P	-	P	-	P	-	P
	REWRITE	-	-	P	-	-	-	P	-
	START	P	-	P	-	-	-	-	-
	DELETE	-	-	P	-	-	-	-	-
RANDOM	OPEN	P	P	P	X				
	READ	P	-	P					
	WRITE	-	P	P					
	REWRITE	-	-	P					
	START	-	-	-					
	DELETE	-	-	P					
DYNAMIC	OPEN	P	P	P					
	READ	P	-	P					
	WRITE	-	P	P					
	REWRITE	-	-	P					
	START	P	-	P					
	DELETE	-	-	P					

P indicates that this input/output statement is permissible for this combination of File Organization, Access Mode and OPEN Mode
 - indicates that this input/output statement is not permissible for this combination of File Organization, Access Mode, and OPEN Mode

Figure IV. OPEN Statement Options and Permissible I/O Statements

A file may be opened for INPUT, OUTPUT, I-O, or EXTEND in the same program. After the first execution of an OPEN statement for a given file, each subsequent execution of an OPEN statement must be preceded by the successful execution of a CLOSE statement without the LOCK option.

Execution of an OPEN INPUT or OPEN I-O statement sets the Current Record Pointer to the first record existing in the file. For indexed files, the record with the lowest key value is considered the first record in the file. If no records exist in the file, the Current Record Pointer is set so that the first Format 1 READ statement executed results in an AT END condition.

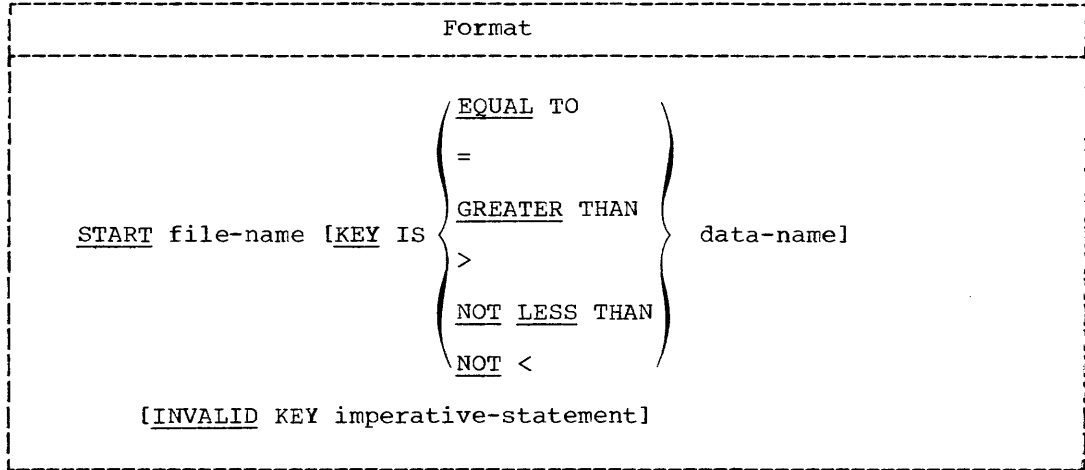
If the PASSWORD clause is specified in the File-Control entry, the password data item must contain the valid password before the OPEN statement is executed. If the valid password is not present, the OPEN statement is unsuccessful.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the OPEN statement is executed.

If an OPEN statement is issued for a file already in the open status, the EXCEPTION/ERROR procedure (if specified) for this file is executed.

START Statement

The START statement provides a means for logical positioning within an indexed file for subsequent sequential retrieval of records.



When the START statement is executed, the associated file must be open in INPUT or I-O mode.

File-name must name an indexed VSAM file with sequential or dynamic access. File-name must be defined in an FD entry in the Data Division. File-name must not be the name of a sort file.

When the KEY option is not specified, the EQUAL TO relational operator is implied. When the START statement is executed, the EQUAL TO comparison is made between the current value in the RECORD KEY and the corresponding key field in the file's records. The Current Record Pointer is positioned to the logical record in the file whose key field satisfies the comparison.

When the KEY option is specified, data-name may be either

- The RECORD KEY for this file, or
- Any alphanumeric data item subordinate to the RECORD KEY whose leftmost character position corresponds to the leftmost character position of the RECORD KEY (that is, a generic key).

When the START statement is executed, the comparison specified in the KEY relational operator is made between data-name and the key field in the file's records. If the operands are of unequal size, the comparison proceeds as if the key field were truncated on the right to the length of the data-name. All other numeric and nonnumeric comparison rules apply. The Current Record Pointer is positioned to the first logical record in the file whose key field satisfies the comparison.

If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, and the position of the Current Record Pointer is undefined. (See "INVALID KEY Condition" in the preceding Common Processing Facilities Section.)

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the START statement is executed.

VSAM READ Statement (DOS/VS)

READ Statement

For sequential access, the READ statement makes available the next logical record from a VSAM file. For random access, the READ statement makes available a specified record from a VSAM file.

```
Format 1  
  
READ file-name [NEXT] RECORD [INTO identifier]  
  
[AT END imperative-statement]
```

```
Format 2  
  
READ file-name RECORD [INTO identifier]  
  
[INVALID KEY imperative-statement]
```

When the READ statement is executed, the associated file must be open in INPUT or I-O mode.

File-name must be defined in an FD entry in the Data Division. File-name must not be the name of a sort file.

The INTO identifier option is described in the preceding Common Processing Facilities Section.

Following the unsuccessful execution of a READ statement, the contents of the associated record area and the position of the Current Record Pointer are undefined.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the READ statement is executed.

FORMAT 1: ^{used} When ACCESS MODE SEQUENTIAL is specified or assumed for a VSAM file, this format must be used. For such files the statement makes available the next logical record from the file. For indexed VSAM files, the NEXT option need not be specified; for sequential VSAM files the NEXT option must not be specified.

When ACCESS MODE DYNAMIC is specified for indexed VSAM files, the NEXT option must be specified for sequential retrieval. For such files, the READ NEXT statement makes available the next logical record from the file.

Before a Format 1 READ statement is executed, the Current Record Pointer must be positioned by the successful prior execution of an OPEN, START, or READ statement. When the Format 1 READ statement is executed, the record indicated by the Current Record Pointer is made available. For sequential VSAM files, the next record is the succeeding record in logical sequence. For a sequentially accessed indexed VSAM file, the next record is that one having the next higher RECORD KEY in collating sequence.

If the position of the Current Record Pointer is undefined when a Format 1 READ statement is issued, the execution of the statement is unsuccessful.

If, when a Format 1 READ statement is executed, no next logical record exists in the file, the AT END condition exists. The execution of the READ statement is considered unsuccessful.

When the AT END condition is recognized, the following actions are taken in the following order:

1. If the FILE-STATUS clause is specified in the File-Control entry, the Status Key is updated to indicate the AT END condition.
2. If the AT END option of the READ statement is specified, control is transferred to the AT END imperative-statement.
3. If the AT END option is not specified, and a USE AFTER EXCEPTION/ERROR procedure for this file is specified, either explicitly or implicitly, the USE procedure is executed.

For files with SEQUENTIAL organization, when the AT END condition has been recognized, a READ statement for this file must not be executed until a successful CLOSE statement followed by a successful OPEN statement have been executed for this file.

For files with INDEXED organization, when the AT END condition is recognized, a Format 1 READ statement for this file must not be executed until one of the following has been successfully executed:

- A CLOSE statement followed by an OPEN statement
- A Format 2 READ statement (dynamic access)
- A START statement

If a sequential VSAM file with the OPTIONAL clause is not present at the time the file is opened, execution of the first READ statement causes the AT END condition to occur. Standard end-of-file procedures are not performed.

FORMAT 2: This format must be used for indexed VSAM files in random access mode, and for random record retrieval in the dynamic access mode.

Execution of a Format 2 READ statement causes the value in the RECORD KEY to be compared with the values contained in the corresponding key field in the file's records until a record having an equal value is found. The Current Record Pointer is positioned to this record, which is then made available.

If no record can be so identified, an INVALID KEY condition exists, and execution of the READ statement is unsuccessful. (See "INVALID KEY Condition" in the preceding Common Processing Facilities Section.)

WRITE Statement

The WRITE statement releases a logical record to an OUTPUT, I-O, or EXTEND file.

Format
<p><u>WRITE</u> record-name [<u>FROM</u> identifier]</p> <p>[<u>INVALID</u> KEY imperative-statement]</p>

~~Sequential Access Mode (SAM)~~

When the WRITE statement is executed, the associated file must be open in OUTPUT, I-O, or EXTEND mode.

Record-name must be the name of a logical record in the File Section of the Data Division. Record-name may be qualified. Record-name must not be associated with a sort file.

The maximum record size for the file is established at the time the file is created, and must not subsequently be changed.

Execution of the WRITE statement releases a logical record to the file associated with record-name.

① After the WRITE statement is executed, the logical record is no longer available in record-name, unless:

- 1) The associated file is named in a SAME RECORD AREA clause (in which case the record is also available as a record of the other files named in the SAME RECORD AREA clause), or
- 2) The WRITE statement is unsuccessful due to a boundary violation.

In either of these two cases, the logical record is still available in record-name.

3) If the FROM identifier option is specified, then after the WRITE statement is executed, the information is still available in identifier, even though it may not be in record-name. (See "INTO/FROM Identifier Option" in the preceding Common Processing Facilities Section.)

★ The Current Record Pointer is not affected by execution of the WRITE statement.

The number of character positions required to store the record in a VSAM file may or may not be the same as the number of character positions defined by the logical description of that record in the COBOL program.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status key is updated when the WRITE statement is executed.

SEQUENTIAL VSAM FILES: The INVALID KEY option must not be specified.

When an attempt is made to write beyond the externally-defined boundaries of the file, the execution of the WRITE statement is unsuccessful, and an EXCEPTION/ERROR condition exists. The contents of record-name are unaffected. If an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is then executed; if no such procedure is specified, the results are undefined.

INDEXED VSAM FILES: Before the WRITE statement is executed, the contents of the RECORD KEY must be set to the desired value. Note that the value contained in any specific RECORD KEY must be unique within the records in the file.

When the WRITE statement is executed, the contents of the RECORD KEY are utilized so that subsequent access to the record can be based on the RECORD KEY.

If sequential access mode is specified or implied, records must be released to the file in ascending order of RECORD KEY.

If random or dynamic access is specified, records may be released in any program-specified order.

INVALID KEY Option: The INVALID KEY condition exists when any of the following conditions occur:

INVALID KEY

VSAM REWRITE Statement (DOS/VS)

- For an OUTPUT or EXTEND file in sequential access mode, when the value of the RECORD KEY is not greater than the value of the RECORD KEY for the previous record.
- For an I-O or OUTPUT file in random or dynamic access mode, when the value of the RECORD KEY is equal to the value of a RECORD KEY for an already existing record.
- When an attempt is made to write beyond the externally-defined boundaries of the file.

When the INVALID KEY condition is recognized, the execution of the WRITE statement is unsuccessful, the contents of record-name are unaffected, and the Status Key, if specified, is set to a value to indicate the cause of the condition. (See "INVALID KEY Condition" and "Status Key" in the preceding Common Processing Facilities Section.)

REWRITE Statement

The REWRITE statement logically replaces an existing record in a VSAM file.

Format
<pre>REWRITE record-name [FROM identifier] [INVALID KEY imperative-statement]</pre>

- ① When the REWRITE statement is executed, the associated file must be open in I-O mode.

Record-name must be the name of a logical record in the File Section of the Data Division. Record-name must not be associated with a sort file. Record-name may be qualified.

- ② Execution of the REWRITE statement replaces an existing record in the file with the information contained in record-name. For a sequential VSAM file, the number of character positions in record-name must equal the number of character positions in the record being replaced. For an indexed VSAM file, the number of character positions in record-name need not equal the number of character positions in the record being replaced.

After successful execution of a REWRITE statement, the logical record is no longer available in record-name unless the associated file is named in a SAME RECORD AREA clause (in which case the record is also available as a record of the other files named in the SAME RECORD AREA clause).

The Current Record Pointer is not affected by execution of the REWRITE statement.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the REWRITE statement is executed.

For files in the sequential access mode, the last prior input/output statement executed for this file must be a successfully executed READ statement. When the REWRITE statement is executed, the record retrieved by that READ statement is logically replaced.

INVALID KEY Statement (DOS/VS)

SEQUENTIAL FILES: The INVALID KEY option must not be specified for this type of file. An EXCEPTION/ERROR declarative procedure may be specified.

INDEXED FILES: For an indexed file in the sequential access mode, the record to be replaced by the REWRITE statement is identified by the current value of the RECORD KEY. When the REWRITE statement is executed, the RECORD KEY must contain the value of the RECORD KEY for the last-retrieved record from the file.

For an indexed file in random or dynamic access mode, the record to be replaced is the record identified by the value of the RECORD KEY.

The INVALID KEY condition exists when:

- The access mode is sequential, and the value contained in the RECORD KEY of the record to be replaced does not equal the RECORD KEY of the last-retrieved record from the file.
- The value contained in the RECORD KEY does not equal that of any record in the file.

If either condition exists, the execution of the REWRITE statement is unsuccessful, the updating operation does not take place, and the data in record-name is unaffected. (See "INVALID KEY Condition" in the preceding Common Processing Facilities Section.)

DELETE Statement

The DELETE statement logically removes a record from an indexed VSAM file.

```
-----  
                        Format  
-----  
  
DELETE file-name RECORD  
  
[INVALID KEY imperative-statement]  
  
-----
```

When the DELETE statement is executed, the associated file must be open in I-O mode.

File-name must be defined in an FD entry in the Data Division and must be the name of an indexed VSAM file.

For a file in sequential access mode, the INVALID KEY option must not be specified.

For a file in random or dynamic access mode, the INVALID KEY option may be specified.

For a file in sequential access mode, the last prior input/output statement must be a successfully executed READ statement. When the DELETE statement is executed, the system logically removes the record retrieved by that READ statement. The current record pointer is not affected by execution of the DELETE statement.

For a file in random or dynamic access mode, when the DELETE statement is executed, the system logically removes the record identified by the contents of the associated RECORD KEY data item. If the file does not contain the record specified by the key, an INVALID KEY condition exists. (See "INVALID KEY Condition" in the preceding Common Processing Facilities section.)

After successful execution of a DELETE statement, the record is logically removed from the file and can no longer be accessed. Execution of the DELETE statement does not affect the contents of the record area associated with file-name.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the DELETE statement is executed.

CLOSE Statement

The CLOSE statement terminates the processing of VSAM files.

Format
<pre>CLOSE file-name-1 [WITH LOCK] [file-name-2 [WITH LOCK]]</pre>

A CLOSE statement may be executed only for a file in an open mode. After successful execution of a CLOSE statement, the record area associated with the file-name is no longer available. Unsuccessful execution of a CLOSE statement leaves availability of the record area undefined.

Each file-name designates a file upon which the CLOSE statement is to operate.

When the WITH LOCK option is not specified, standard system closing procedures are performed. This file may be opened again during this execution of the object program.

When the WITH LOCK option is specified, standard system closing procedures are performed; the compiler ensures that this file cannot be opened again during this execution of this object program.

After a CLOSE statement is successfully executed for the file, an OPEN statement for that file must be executed before any other input/output statement can refer explicitly or implicitly to the file.

If a CLOSE statement is not executed for an open file before a STOP RUN statement is executed, results are unpredictable.

If an input sequential VSAM file is described in the File-Control entry as OPTIONAL and the file is not present during this execution of the object program, standard end-of-file processing is not performed.

If the FILE STATUS clause is specified in the File-Control entry, the associated Status Key is updated when the CLOSE statement is executed.

If the file is in an open status and the execution of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedure (if specified) for this file is executed.

MERGE FACILITY

The Merge Facility gives the COBOL user access to the merging capabilities of the Program Product DOS/VS Sort-Merge (Program Number 5746-SM1). Through COBOL, the user can combine two or more identically

SELECT/ASSIGN for MERGE (DOS/VS)

ordered input files into one output file according to key(s) contained in each record. More than one merge operation can be performed during one execution of the COBOL program. Special processing of output records can also be specified.

There are special considerations in the Environment Division, the Data Division, and the Procedure Division for the Merge Facility.

ENVIRONMENT DIVISION

Each input file and the resulting merged output file must be described in a separate File-Control entry, and each must be a standard sequential file, or a VSAM file with sequential access. The merge file must have a separate File-Control entry, as described in the following paragraphs.

File-Control Entry for Merge Files

The File-Control entry names the merge file and associates it with a storage medium.

General Format
{ <u>SELECT</u> file-name
<u>ASSIGN</u> TO system-name-1 [system-name-2]}...

Each File-Control entry for a merge file must begin with a SELECT clause, and be immediately followed by an ASSIGN clause. There may be no other clauses.

SELECT Clause: The SELECT clause names each merge file in the program. Each file described by an SD entry in the Data Division must be named once and only once as a file-name following the key word SELECT.

ASSIGN Clause: The ASSIGN clause is required. System-name has the same rules of formation as it has for sort work files; however, the fixed SYSnnn and name fields (SYS001 and SORTWK1, etc.) are treated as documentation. (See "Assignment of Sort Work Units" in the Sort Feature chapter.)

If an ASCII-collated merge is to be performed, C must be specified in the organization field. (See "Appendix E: ASCII Considerations.")

I-O-Control Paragraph

The optional I-O-Control Paragraph specifies the storage area to be shared by different files.

General Format		
SAME	$\left. \begin{array}{l} \text{SORT} \\ \text{SORT-MERGE} \\ \text{RECORD} \end{array} \right\}$	AREA FOR
file-name-1 [file-name-2]		

When the SAME SORT AREA or SORT-MERGE AREA clause is specified, at least one file-name specified must name a sort or merge file. Files that are not sort or merge files may also be specified. The following rules apply:

- More than one SAME SORT AREA or SORT-MERGE AREA clause may be specified; however, a sort or merge file must not be named in more than one such clause.
- If a file that is not a sort or merge file is named in both a SAME AREA clause and in one or more SAME SORT AREA or SORT-MERGE AREA clauses, all of the files in the SAME AREA clause must also appear in all of the SAME SORT AREA or SORT-MERGE AREA clauses.
- Files named in a SAME SORT AREA or SORT-MERGE AREA clause need not have the same organization or access.
- Files named in a SAME SORT AREA or SORT-MERGE AREA clause that are not sort or merge files do not share storage with each other unless the user names them in a SAME AREA or SAME RECORD AREA clause.

The SAME SORT AREA or SORT-MERGE AREA clause specifies one storage area available for merge operations by each named merge file. That is, the storage area allocated for one merge operation is available for reuse in another merge operation.

The function of the SAME SORT AREA or SORT-MERGE AREA clause is to optimize the assignment of storage areas to a given MERGE statement. The system handles storage assignment automatically; hence, the SORT AREA or SORT-MERGE AREA options, if specified, are treated as documentation.

When the SAME RECORD AREA option is specified, the named files, including any sort or merge files, share only the area in which the current logical record is processed. Several of the files may be open at the same time, but the logical record of only one of these files can exist in the record area at one time.

DATA DIVISION

In the Data Division, the user must include file description entries for each merge input file and for the merged output file, merge-file description entries for each merge file, and record description entries for each.

Merge-File Description Entry

A merge-file description entry must appear in the File Section for each merge file named in a File-Control entry.

```
General Format

SD merge-file-name

[RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]

[DATA {RECORD IS } data-name-1 [data-name-2] ...]
      {RECORDS ARE }

[SORT-OPTION IS data-name-3].
```

The level indicator SD identifies the beginning of the merge-file description, and must precede the merge-file-name.

The clauses following merge-file-name are optional, and their order of appearance is not significant.

One or more record description entries must follow the merge-file description entry, but no input/output statements may be executed for the merge file.

Merge-File-Name: The merge-file-name must be the same as that specified in the merge file File-Control entry. It is also the name specified as the first operand in the MERGE statement.

RECORD CONTAINS Clause: The size of each data record is completely defined in the record description entry; therefore, this clause is never required. When it is specified, the same considerations apply as in its Data Division description.

DATA RECORDS Clause: This clause names the 01-level data records associated with this SD entry. This clause is never required, and the compiler treats it as documentation. When it is specified, the same considerations apply as in its Data Division description.

SORT-OPTION Clause: This clause specifies that at object time an OPTION control statement for the Sort/Merge program will be specified in the data-name-3 area. Rules for specification are given in the DOS/VS Sort Feature description later in this chapter.

PROCEDURE DIVISION

The Procedure Division contains a MERGE statement describing the merge operation, and optional output procedures. The procedure-names of the output procedures are specified within the MERGE statement. More than one MERGE statement can be specified, appearing anywhere except in the declaratives section or in an input or output procedure for a SORT or MERGE statement.

MERGE Statement

The MERGE statement combines two or more identically sequenced files using specified key(s), and makes records available to an output file in merged order.

```

#                                     Format
MERGE file-name-1
      ON { ASCENDING } KEY data-name-1 [data-name-2] ...
         { DESCENDING }
      [ON { ASCENDING } KEY data-name-3 [data-name-4] ... ] ...
         { DESCENDING }
      USING file-name-2 file-name-3 [file-name-4] ...
      { GIVING file-name-5
        OUTPUT PROCEDURE IS section-name-1 [THRU section-name-2] }

```

No file-name specified in the MERGE statement may be open at the time the statement is executed. The files are automatically opened and closed by the merge operation; all implicit functions are performed, such as execution of system procedures or any associated declarative procedures.

No file-name may be specified more than once in one MERGE statement.

Only one file-name from a multiple file reel may appear in one MERGE statement.

FILE-NAME-1: This file-name represents the merge file, and must be described in an SD entry in the Data Division.

ASCENDING/DESCENDING KEY Option: These options specify whether records are to be merged in ascending or descending sequence, based on one or more merge keys.

Each data-name represents a KEY data item, and must be described in the record description(s) associated with the SD entry for file-name-1, the merge work file. The following rules apply:

- if file-name-1 has more than one associated record description entry, the KEY data items need be described in only one such record description
- each data-name may be qualified; it may not be subscripted or indexed (that is, it may not contain or be contained in an entry that contains an OCCURS clause)
- KEY data items must be at a fixed displacement from the beginning of the record (that is, no KEY data item may follow an OCCURS DEPENDING ON clause in the record description)
- a maximum of 12 keys may be specified; the total length of all keys must not exceed 256 bytes
- all key fields must be located within the first 4092 bytes of the logical record

The KEY data items are listed in order of decreasing significance, no matter how they are divided into KEY phrases. Using the format as an example, data-name-1 is the most significant key, and records are merged in ascending or descending order on that key; data-name-2 is the next most significant key; within data-name-1, records are merged on

data-name-2 in ascending or descending order. Within data-name-2, records are merged on data-name-3 in ascending or descending order; within data-name-3, records are merged on data-name-4 in ascending or descending order, etc.

When ASCENDING is specified, the merged sequence is from the lowest to the highest value of the contents in the KEY data item according to the collating sequence used.

When DESCENDING is specified, the merged sequence is from the highest to the lowest value of the contents in the KEY data item according to the collating sequence used.

Figure V gives the collating sequence used for each category of KEY data item.

KEY Category	Collating Sequence
Alphabetic Alphanumeric Alphanumeric Edited Numeric Edited	EBCDIC (non-algebraic and unsigned)
Numeric	Algebraic (signed)

Figure V. KEY Item Categories and Collating Sequences

The rules for comparison are those for the relation condition (see "Relation Condition" in the Conditions chapter of the Procedure Division). If two or more KEY data items test as equal, the merge operation makes the records available in the order that the input file-names are specified in the USING option.

USING Option: All file-names listed in the USING option represent identically ordered input files that are to be merged. Two through eight file-names may be specified.

GIVING Option: File-name-5 is the name of the merged output file. When this option is specified, all merged records made available from the merge operation are automatically written on the output file.

OUTPUT PROCEDURE Option: When this option is specified, all output records from the merge operation are made available to the user (through a RETURN statement) for further processing.

When an output procedure is specified, control passes to the procedure during execution of the MERGE statement. Before entering the output procedure, the merge operation reaches a point at which it can provide the next merged record when requested. The RETURN statement in the output procedure is a request for the next merged record. (See the RETURN statement description in the Sort Feature chapter.) An output procedure must contain at least one RETURN statement to make merged records available for further processing.

Control may be passed to an output procedure only when a related MERGE statement is being executed.

The output procedure must not form part of any other procedure.

If section-name-1 alone is specified, the output procedure must consist of one contiguous Procedure Division section.

If section-name-1 THRU section-name-2 is specified, the output procedure consists of two or more contiguous Procedure Division sections; section-name-1 specifies the first such section; section-name-2 specifies the last such section.

Control must not be passed to the output procedure unless a related MERGE or SORT statement is being executed, because RETURN statements in the output procedure have no meaning unless they are controlled by a MERGE or SORT statement. The output procedure may consist of the processing requests necessary to select, modify, or copy the records being made available, one at a time, from the merge operation. The following restrictions apply:

- There may be no explicit transfers of control outside the output procedure. ALTER, GO TO, and PERFORM statements within the procedure must not refer to procedure-names outside the output procedure. However, an implicit transfer of control to a declarative procedure is allowed.
- No SORT or MERGE statements are allowed.
- The remainder of the Procedure Division must not transfer control to points inside the output procedure; that is, ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division must not specify procedure-names within an output procedure.

The compiler inserts an end-of-processing transfer at the end of the last output procedure section. When end-of-processing is recognized, the merge operation is terminated, and control is transferred to the next statement following the MERGE statement.

SEGMENTATION RESTRICTIONS: The MERGE statement may be specified in a segmented program. However, the following restrictions apply:

- If the MERGE statement appears in the fixed portion, then any associated output procedure must be:
 - completely within the fixed portion, or
 - completely within one independent segment
- If the MERGE statement appears in an independent segment, then any associated output procedure must be:
 - completely within the fixed portion, or
 - completely within the same independent segment as the MERGE statement

3886 OCR PROCESSING

Document reader

The IBM 3886 OCR (Optical Character Reader) Model 1 is a general purpose online unit record device that satisfies a broad range of data entry requirements. The 3886 OCR can significantly reduce time and cost factors, by eliminating input steps in both new and existing applications; a keying process is no longer necessary, since the 3886 OCR can read and recognize data created by numeric hand printing, high-speed computer printing, typewriters, and preprinted forms.

The IBM 3886 OCR uses several new technologies which make it a compact, highly reliable, modular device. A powerful microprogrammed recognition and control processor performs all machine control and character recognition functions, and enables the 3886 OCR to perform sophisticated data and blank editing.

The 3886 OCR accepts documents from 3 x 3 to 9 x 12 inches in size. Under program control, it can read documents line-by-line, transmitting their contents line-by-line to the CPU. Additional facilities, all under program control, include: document marking, line marking, document ejecting (with stacker selection), and line reading (of current line).

FIPS FLAGGER (DOS/VS)

DOS/VS COBOL support for the 3886 OCR is through an object-time subroutine in the COBOL library, invoked through COBOL CALL statements. By means of parameters passed to the subroutine, the following operations are provided: open and close the file, read a line, wait for read completion, mark a line, mark the current document, eject the current document, and load a format record. After each operation, a status indicator is passed back to the COBOL program, so that any exceptional condition can be tested.

Through a fixed format OCR file information area in the Working-Storage or Linkage Section, the COBOL user defines storage for the OCR parameters. Of these parameters, the COBOL programmer is responsible for providing a file identifier, a format record identifier, an operation code, and (depending on the operation) a line number, line format number, mark code, and stacker number. After completion of each operation a status indicator is returned; after completion of a read operation, header and data records are also returned.

DOS/VS provides two macro instructions for defining documents. The DFR macro instruction defines attributes common to a group of line types. The DLINT macro instruction defines specific attributes of an individual line type. The DFR and associated DLINT macro instructions are used in one assembly to build a format record module. The format record must be link-edited into the core image library so that it can be loaded into the 3886 OCR when the file is to be processed. The format record indicates the line types to be read, attributes of the fields in the lines, and the format of the data records to be processed.

Additional information on the IBM 3886 OCR can be found in the following publications:

IBM DOS/VS COBOL Compiler and Library Programmer's Guide, Order No. SC28-6478

IBM 3886 Optical Character Reader

General Information Manual, Order No. GA21-9146

Input Document Design Guide and Specifications, Order No. GA21-9148

DOS/VS Program Planning Guide for IBM 3886 Optical Character Reader Model 1, Order No. GA21-5099

DOS/VS Supervisor and I/O Macros, Order No. GC33-5373

FIPS FLAGGER

The FIPS (Federal Information Processing Standard) is a compatible subset of full American National Standard COBOL, X3.23-1968. The FIPS itself is subdivided into four levels: low, low-intermediate, high-intermediate, and full. Any program written to conform to the FIPS must conform to one of those levels of FIPS processing. Processing modules included in Full American National Standard COBOL, and those included in the four levels of the FIPS, are shown in Figure VI.

American National Standard COBOL Processing Modules	Full FIPS Processing Modules	High-intermediate FIPS Processing Modules	Low-intermediate FIPS Processing Modules	Low FIPS Processing Modules
2NUC 1,2 (Nucleus)	2NUC 1,2	2NUC 1,2	2NUC 1,2	1NUC 1,2
3TBL 1,3 (Table Handling)	3TBL 1,3	2TBL 1,3	2TBL 1,3	1TBL 1,3
2SEQ 1,2 (Sequential Access)	2SEQ 1,2	2SEQ 1,2	2SEQ 1,2	1SEQ 1,2
2RAC 0,2 (Random Access)	2RAC 0,2	2RAC 0,2	2RAC 0,2	
2SRT 0,2 (Sort)	2SRT 0,2	1SRT 0,2		
2RPW 0,2 (Report Writer)				
2SEG 0,2 (Segmentation)	2SEG 0,2	1SEG 0,2	1SEG 0,2	
2LIB 0,2 (Library)	2LIB 0,2	1LIB 0,2	1LIB 0,2	

Figure VI. The Four Levels of FIPS Processing

The FIPS Flagger identifies source clauses and statements that do not conform to the Federal standard. Four levels of flagging, to conform to the four levels of the FIPS, are provided. The following lists identify COBOL source elements flagged for each level.

FULL FIPS FLAGGING: When flagging for the full FIPS level is specified, the following elements of the COBOL source, if specified, are identified.

GLOBAL ITEMS

Single quote instead of double
Floating Point Literals

- Special Register LINE-COUNTER
- Special Register PAGE-COUNTER
- Special Register CURRENT-DATE
- Special Register TIME-OF-DAY
- Special Register COM-REG
- Special Register SORT-RETURN
- Special Register SORT-FILE-SIZE
- Special Register SORT-CORE-SIZE
- Special Register SORT-MODE-SIZE
- Special Register NSTD-REELS
- Special Register WHEN-COMPILED

Comment Lines with * in Column 7
The SUPPRESS option of the COPY statement

FIPS Flagger (DOS/VS)

IDENTIFICATION DIVISION Items

ID abbreviation for IDENTIFICATION
Accepting Identification Division Paragraphs in any order
Accepting Program Name in quotes

ENVIRONMENT DIVISION Items

Optional CONFIGURATION SECTION and Paragraphs
S01 through S05 Function-names in SPECIAL-NAMES paragraph

Allowing any order for optional SELECT clauses
W, R, or I as Organization indicator in System-name
Optional omission of IS in ACCESS MODE IS Clause
Optional omission of IS in ACTUAL KEY IS Clause
ACTUAL-KEY clause for sequential access of a direct file
ACTUAL-KEY clause for sequential creation of a direct File
NOMINAL KEY Clause in FILE-CONTROL Paragraph
RECORD KEY Clause in FILE-CONTROL Paragraph
TRACK-AREA Clause in FILE-CONTROL Paragraph
The COPY statement in the FILE-CONTROL paragraph

Short form of RERUN ON Clause
Interchangeable use of REEL and UNIT in RERUN ON Clause
APPLY Clause in I-O-CONTROL paragraph
Allowing I-O-CONTROL paragraph clauses in any order

RESERVE integer AREAS clause (as distinguished from the RESERVE
ALTERNATE AREAS clause)
ORGANIZATION clause
ACCESS MODE DYNAMIC clause
PASSWORD clause
FILE STATUS clause
SAME SORT-MERGE AREA clause

DATA DIVISION Items

REPORT SECTION of DATA DIVISION
RD level indicator
The DATA RECORDS clause for a REPORT FD
LINKAGE SECTION of DATA DIVISION

Allowing unequal level numbers to belong to the same group
RECORDING MODE Clause of FD entry.
REPORT Clause of FD Entry
LABEL RECORDS CLAUSE on Sort File Description
SORT-OPTION clause on Sort File Description
Optional BLOCK CONTAINS for DIRECT Files when RECORDING MODE IS S
Accepting name of preceding entry when using multiple redefinition
External Floating-point picture
The SIGN Clause
Allowing the SYNCHRONIZED Clause at the 01 level
COMPUTATIONAL-1 option of the USAGE Clause
COMPUTATIONAL-2 option of the USAGE Clause
COMPUTATIONAL-3 option of the USAGE Clause
COMPUTATIONAL-4 option of the USAGE Clause
Nested OCCURS DEPENDING ON clauses
Allowing SYNCHRONIZED with USAGE IS INDEX
The COPY statement in the Working-Storage Section
DISPLAY-ST option of the USAGE Clause and associated PICTURE
Use of VALUE Clause as Comments in File Section for other than
Condition-name entries
COPY REDEFINES in Working-Storage Section

PROCEDURE DIVISION Items

USING clause on PROCEDURE DIVISION
 THEN used to separate statements
 Allowing omission of section header at beginning of Procedure
 Division
 The START statement
 The REWRITE statement
 The TRANSFORM statement

The GENERATE statement
 The INITIATE statement
 The TERMINATE statement

The DEBUG statement
 The READY TRACE statement
 The RESET TRACE statement
 The ON statement
 The EXHIBIT statement
 The CALL statement
 The ENTRY statement
 The GOBACK statement
 The EXIT PROGRAM statement

The USE AFTER STANDARD EXCEPTION sentence
 The READ NEXT statement
 The DELETE statement
 The MERGE statement
 The EXTEND option for the OPEN statement and Error Procedures
 The SERVICE RELOAD statement

The unary plus operator
 Allowing omission of the space following the unary operator
 OTHERWISE in IF statements
 The GO TO MORE-LABELS statement
 GIVING option of USE sentence
 USE BEFORE REPORTING sentence
 Allowing omission of the INVALID KEY option for READ and WRITE
 statements
 The AT END-OF-PAGE or EOP option of the WRITE statement
 The WRITE AFTER POSITIONING statement
 The FROM SYSIPT or CONSOLE option of the ACCEPT statement
 The UPON CONSOLE, SYSPUNCH, SYSPCH, or SYSLST option of the DISPLAY
 statement

The BASIS statement
 The INSERT statement
 The DELETE statement

The EJECT statement
 The SKIP1 statement
 The SKIP2 statement
 The SKIP3 statement

HIGH-INTERMEDIATE FIPS FLAGGING: When flagging for the high-intermediate FIPS level is specified, all elements included in the preceding list are flagged, plus the following additional COBOL source elements:

GLOBAL ITEMS

The REPLACING option of the COPY statement

FIPS Flagger (DOS/VS)

ENVIRONMENT DIVISION

SEGMENT-LIMIT clause in OBJECT-COMPUTER paragraph
SORT option of SAME Clause

DATA DIVISION

The ASCENDING and DESCENDING KEY option of the OCCURS clause
The DEPENDING ON option of the OCCURS clause

PROCEDURE DIVISION

All sections with the same priority number must be together
All segments with priority number 1-49 must be together

The SEARCH statement
More than one SORT statement
The FROM option of the RELEASE statement
The INTO option of the RETURN statement

LOW-INTERMEDIATE FIPS FLAGGING: When flagging for the low-intermediate FIPS level is specified, all elements included in the preceding lists are flagged, plus the following additional COBOL source elements:

ENVIRONMENT DIVISION

The OR option of the SELECT sentence

DATA DIVISION

SD level indicator

PROCEDURE DIVISION

One or more SORT statements
Only one STOP RUN statement in the non-declarative portion
The RETURN statement
The RELEASE statement

LOW FIPS FLAGGING: When flagging for the low FIPS level is specified, all elements included in the preceding lists are flagged, plus the following additional COBOL source elements:

GLOBAL ITEMS

Comma and semicolon as punctuation
Datanames which begin with non-alphabetic character
Continuation of words and numeric literals

Figurative constant ZEROES
Figurative constant ZEROS
Figurative constant SPACES
Figurative constant HIGH-VALUES
Figurative constant LOW-VALUES
Figurative constant QUOTES
Figurative constant ALL literal
The COPY statement

IDENTIFICATION DIVISION

DATE-COMPILED Paragraph

ENVIRONMENT DIVISION

RESERVE ALTERNATE AREAS Clause in File-Control Paragraph (SELECT sentence)
 OPTIONAL in SELECT Clause
 ACTUAL KEY Clause in File-Control Paragraph
 FILE-LIMITS ARE Clause
 Data-name instead of literal in FILE-LIMIT IS clause
 Multiple extents in FILE-LIMIT IS clause
 RANDOM option in ACCESS MODE IS Clause

 RECORD and file-name-2 option of SAME Clause
 MULTIPLE FILE TAPE Clause in I-O-CONTROL Paragraph

DATA DIVISION

Level numbers 11 - 49
 Level numbers 1 - 9 (1-digit)
 Level number 66 RENAMES clause
 Level number 88 Condition Name
 Nesting of REDEFINES Clause
 VALUE Clause as Condition-name entry
 Integer-1 TO option of BLOCK CONTAINS (RECORD or CHARACTER) Clause
 Data-name option on LABEL RECORDS Clause
 Data-name option of VALUE OF Clause
 Multiple Index-names for OCCURS clause

PROCEDURE DIVISION

+, -, *, /, and **
 >, <, and = in relationals
 Connectives OF, IN, ', ", AND, OR, and NOT
 DECLARATIVES, END DECLARATIVES and USE sentence
 Qualification of names
 Priority number on Section header
 The COMPUTE verb
 The SEEK Statement
 The Sign condition (POSITIVE, NEGATIVE, or ZERO)
 Condition-name condition
 Compound conditions
 Nested IF statements
 CORRESPONDING option (ADD, SUBTRACT, and MOVE)
 Multiple results of ADD and SUBTRACT statements
 REMAINDER option of DIVIDE statement
 GO TO without object (used with ALTER)
 Multiple operands of ALTER statement
 UNTIL Condition and VARYING form of PERFORM
 REVERSED and NO REWIND options of OPEN statement
 Multiple file-names in OPEN statement
 INTO option of READ statement
 INVALID KEY option of READ statement
 FROM option of WRITE statement
 ADVANCING identifier LINES/mnemonic/name form of WRITE
 The FROM option of the ACCEPT statement
 The UPON option of the DISPLAY statement
 The WITH NO REWIND or LOCK option of the CLOSE statement
 Multiple file-names in a CLOSE statement
 Three levels of subscripting
 Multiple Index-names/identifier in SET statement
 The UP BY and DOWN BY option of the SET statement

DOS/VS Devices (DOS/VS)

MISCELLANEOUS FILE PROCESSING CONSIDERATIONS

The following items, concerning standard sequential, direct, and indexed file processing, as well as the sort feature, apply only for DOS/VS COBOL.

File Processing Summary -- New Devices

The file processing techniques available for the DOS/VS COBOL devices are summarized in Figure VII.

DOS Organization	Device	ACCESS	Organization
DTFCD	5425	[SEQUENTIAL]	Standard Sequential
DTFPR	3203/5203, 5425	[SEQUENTIAL]	Standard Sequential
DTFDU	3540	[SEQUENTIAL]	Standard Sequential
DTFSD	3340	[SEQUENTIAL]	Standard Sequential
DTFDA	3340	[SEQUENTIAL]	direct
DTFDA	3340	RANDOM	direct
DTFIS	3340	[SEQUENTIAL]	indexed
DTFIS	3340	RANDOM	indexed

Figure VII. File Processing Summary -- DOS/VS COBOL Devices

ASSIGN Clause

For the new DOS/VS COBOL devices, system/name has the following formats:

For the 3203/5203 printers:

$$\text{SYSnnn-UR-} \left\{ \begin{array}{l} 3203 \\ 5203 \end{array} \right\} \text{-S[-name]}$$

For the 5425 multifunction card unit:

$$\text{SYSnnn-UR-5425} \left\{ \begin{array}{l} P \\ R \\ W \end{array} \right\} - \left\{ \begin{array}{l} S \\ T \\ V \\ X \\ Y \\ Z \end{array} \right\} \left\{ \begin{array}{l} [P] \\ S \end{array} \right\} \text{ [-name]}$$

(See the ASSIGN Clause description for the 2560 MFCM for the meaning of each field.)

For the 3540 diskette input/output unit:

$$\text{SYSnnn-} \left\{ \begin{array}{c} \text{UT} \\ \text{DA} \end{array} \right\} \text{-3540-S[-name]}$$

For the 3340 mass storage disk facility:

$$\text{SYSnnn-} \left\{ \begin{array}{c} \text{UT-3340-S} \\ \text{DA-3340-} \left\{ \begin{array}{c} \text{S} \\ \text{A} \\ \text{D} \\ \text{U} \\ \text{W} \\ \text{I} \end{array} \right\} \end{array} \right\} \text{[-name]}$$

ACTUAL KEY Clause

When the 3340 device is specified for a direct file, and actual track addressing is used, the first 8 bytes of the ACTUAL KEY may be specified as follows:

	PACK	CELL		CYLINDER		HEAD		RECORD
	M	B	B	C	C	H	H	R
Byte	1	2	3	4	5	6	7	8
Device								
3340 (Mod 35)	0-221	0	0	0-347		0-11		0-255
3340 (Mod 70)	0-221	0	0	0-695		0-11		0-255

APPLY WRITE-VERIFY Clause

For the 3540 diskette input/output unit, this clause has no meaning, and must not be specified.

APPLY CYL-OVERFLOW Clause

For a 3340 mass storage facility, when the APPLY CYL-OVERFLOW clause is specified, the maximum number of tracks that can be reserved for overflow records is 10. If the clause is not specified, 2 tracks are reserved for overflow records.

BLOCK CONTAINS Clause

For a file on the 3540 diskette input/output unit, when the BLOCK CONTAINS clause is specified, the RECORDS option must be used.

LABEL RECORDS Clause

For a file on the 3540 diskette input/output unit, LABEL RECORDS ARE STANDARD must be specified.

ERROR Declarative

When the GIVING option is specified for a file on the 3540 diskette input/output unit, the error bytes in data-name-1 contain the information shown in Figure VIII. (For the 3340 device, the contents of the error bytes are the same as for other mass storage files.)

Miscellaneous Considerations (DOS/VS)

Error Byte	Meaning
1	data check
2	equipment check
3 thru 8	unused

Figure VIII. Error Bytes for 3540 -- GIVING Option

ACCEPT Literal Statement

When FROM CONSOLE is specified, the input data can be typed in either capital or small letters, or a combination of the two. The program accepts the data as capital letters.

OPEN Statement

A file that resides on the 3540 diskette input/output unit may be opened only in the INPUT or OUTPUT mode; the REVERSED and WITH NO REWIND options may not be specified.

WRITE Statement

For a file on the 3540 diskette input/output unit, the INVALID KEY option must not be specified.

For a file on the 5425 MFCU, the System/370 Card Device considerations as described in the WRITE Statement of the Procedure Division chapter apply, with the following additional rules:

- For the print feature, the ADVANCING/POSITIONING options are not allowed; single spacing is automatically provided.
- For the print feature, there may be only one WRITE statement issued for each card.
- The print feature allows a maximum of 32 characters per line and 3 or 4 lines per card, for a maximum of 128 characters.
- For the punch feature, in the WRITE AFTER ADVANCING statement for stacker selection, function-names S01 through S04 may be specified.
- For the punch feature, in the WRITE AFTER POSITIONING statement for stacker selection, V, W, X, Y may be specified for stackers 1 through 4, respectively.

CLOSE Statement

For a file on the 3540 diskette input/output unit, only the CLOSE file-name and CLOSE file-name WITH LOCK options are valid.

When the CLOSE statement is executed, standard close file procedures are performed, and the diskette is fed out of the input/output unit. When the WITH LOCK option is specified, the compiler ensures that this file cannot be opened again during this execution of the object program.

SORT-OPTION Clause/5425 Processing (DOS/VS)Sort Feature

The input and/or output file can be either a standard sequential file or a sequentially accessed VSAM file. Up to 8 input files may be specified in the USING option.

The sort-file-description entry may be specified as follows:

Format	
<u>SD</u>	sort-file-name
	[RECORDING MODE IS mode]
[<u>DATA</u>	{ <u>RECORD</u> IS } { <u>RECORDS</u> ARE } data-name-1 [data-name-2] ...]
	[<u>RECORD</u> CONTAINS [integer-1 TO] integer-2 CHARACTERS]
[<u>LABEL</u>	{ <u>RECORD</u> IS } { <u>STANDARD</u> } { <u>RECORDS</u> ARE } { <u>OMITTED</u> }]
	[<u>SORT-OPTION</u> IS data-name-3].

The SORT-OPTION clause specifies that at object time an OPTION control statement for the Sort/Merge program will be specified in the data-name-3 area.

Data-name-3 must be a field defined in the WORKING-STORAGE section. A full description of the SORT-OPTION clause can be found in the DOS/VS COBOL Programmer's Guide, SC28-6478.

The other clauses of the sort-file-description entry are implemented as described in the Sort Feature chapter.

5425 Combined Function Processing

The descriptions in Appendix G apply, with the following special considerations:

In the SPECIAL-NAMES paragraph, function-names S01 through S04 may be specified for stacker selection of 5425 punched output.

ASSIGN clause considerations for the 2560 MFCM apply also for the 5425 MFCU.

For the 5425 print function WRITE statement, line control may not be specified, and there may be only one WRITE statement issued per card. A maximum of 32 characters per line, and 3 or 4 lines per card may be specified, for a maximum logical record size of 128 characters.

For the punch function, S01 through S04 (for stacker selection 1 through 4, respectively) may be specified in the WRITE ADVANCING statement.

For the punch function, V, W, X, Y (for stacker selection 1 through 4, respectively) may be specified in the WRITE AFTER POSITIONING statement.

FEATURES OF THE DOS FULL COBOL PROGRAM PRODUCT COMPILERS

DOS/VS COBOL: This Program Product Compiler and Library includes the following features:

Release 2: The following features are included in this release:

- Lister Facility -- produces a reformatted COBOL source listing and, optionally, a reformatted source deck reflecting the listing. The reformatted source listing contains embedded cross-reference information indicating the statement to or from which the reference is made, and the type of action resulting from the reference. The utility of the listing is increased further by imposing uniform indenting conventions so that the hierarchy of group data items and nested IF statements is made more visible.
- Verb Profiles -- facilitates identifying and locating verbs in the COBOL source program. A summary listing of each different verb and the number of times it occurs is produced. The listing can also include the statement number associated with each occurrence.
- Verb Statistics -- summarizes how often each verb in the COBOL source program is executed during an individual program execution. The statistics are printed at program termination, and are useful in identifying heavily-used portions of a program so that they may be analyzed for optimization. In addition, the statistics serve as verification that a given portion of a program has been executed.
- SORT-OPTION Clause -- In the sort-file-description entry, the SORT-OPTION clause specifies a Working-Storage data area which at object time will contain an OPTION control statement for the Sort/Merge program (Program Number 5746-SM1). The SORT-OPTION clause gives the COBOL programmer greater control over operation of the Sort/Merge Program.

Base Compiler Features: the following features are continued from Release 1.

- VSAM (Virtual Storage Access Method) Support -- which provides fast storage and retrieval of records, password protection, centralized and simplified data and space management, advanced error recovery facilities, plus system catalogs. COBOL supports indexed (key-sequenced) files and sequential (entry-sequenced) files. Records can be fixed or variable in length.
- MERGE Support -- the MERGE verb allows the COBOL user to combine two or more identically ordered input files into one output file according to embedded key(s) in the record, through the Program Product DOS/VS Sort-Merge (Program Number 5746-SM1). Special processing of merged records can also be specified. Both standard sequential files and sequentially accessed VSAM files can be designated as input or output.
- Added System/370 Device Support -- including the following devices:
 - (1) 5425 Multifunction Card Unit (MFCU) -- for 96-column cards, with read/punch/print/select features and combined function processing. Without the combined function features, the 5425 MFCU can be used as a backup reader or as a punch.

- (2) 3203 Advanced Printer -- with 132-character print line
- (3) 3886 Optical Character Reader (OCR) -- which reads multiline alphanumeric or numeric machine-printed documents, or numeric hand-printed documents, with stacker selection.
- (4) 3340 Disk Facility -- high-speed, and large capacity disk.
- (5) 3540 Diskette Input/Output Unit -- transfers IBM Diskette data directly to the System/370 and transfers System/370 data directly to IBM Diskettes. (Data for initial entry via Diskette is prepared using the IBM 3740 Data Entry System.)
- (6) 5203 Advanced Printer -- with 120/132 character print line at compile time, and 96/120/132-character print line at object time.
- FIPS (Federal Information Processing Standard) Flagger -- which issues messages identifying nonstandard elements in a COBOL source program. The FIPS Flagger makes it possible to ensure that COBOL clauses and statements in a DOS/VS COBOL source program conform to the Federal Information Processing Standard.
- WHEN-COMPILED Special Register -- the WHEN-COMPILED special register makes available to the object program the date-and-time compiled constant carried in the object module. WHEN-COMPILED provides a means of associating a compilation listing with both the object program and the output produced at execution time.

All of the features of DOS Full American National Standard COBOL Version 3 continue to be supported by IBM DOS/VS COBOL. (See the following section.) The IBM DOS/VS COBOL Compiler and Library is packaged as a single Program Product, Program Number 5746-CE1; the Library is also available as a separate Program Product, Program Number 5746-LM4.

DOS FULL AMERICAN NATIONAL STANDARD COBOL VERSION 3: this Program Product Compiler includes the following features:

Release 3: The following features are included in this release:

- Additional System/370 Device Support -- including the following:
 - (1) 2560 Multifunction Card Machine (MFCM) -- for 80-column cards. Read/punch/print/select features, and combined function processing are supported. Without the combined function processing feature, the 2560 MFCM can be used as a backup reader or as a punch.
 - (2) 3504 Reader with OMR (Optical Mark Read) feature -- the compiler can use the 3504 without the OMR feature as the SYSIPT device.
 - (3) 3881 Optical Mark Reader (OMR) -- which reads hand-written or machine-printed marks on paper documents. When equipped with the optional BCD feature, the 3881 OMR can also read binary coding.
- Generic Key Facility for Indexed Files -- sequential record retrieval can be requested using a search argument comprised of a user-specified number of high-order characters (generic portion) of the NOMINAL KEY. The user need not specify a full or exact search key.

- Enhanced Compiler Output -- including date, start-time of compilation and program-id on every source listing page. Compiler statistics are also available, and the date and time of compilation are carried as constants in the object module, so that the object module can be associated with an output listing.
- Maintainability Improvements -- the installation can set the compiler default options by cataloging them into the source statement library.

Base Compiler Features: the following features are continued from Release 1 and Release 2.

- Improvements in Object Code to save main storage:
 - (1) Optimized Object Code -- which results, when specified, in up to 30% space saving in object program generated code and global tables as compared with Version 2. The space saved depends on the number of referenced procedure-names and branches, and on 01-level data names.
 - (2) System/370 Support can be requested, to take advantage of the System/370 instruction set. When such support is requested, System/370 instructions particularly suited to COBOL programming are generated to replace the equivalent object-time subroutines and instructions needed when running under System/360. The System/370 instructions save up to 12% of generated object program space, plus the space no longer needed by the subroutines.
 - (3) Double-Buffered ISAM -- allows faster sequential processing of indexed files.
 - (4) Improvements in the MOVE Statement and in Comparisons -- when a MOVE statement or a comparison involves a one-byte literal, generated code for the move and the comparison has been improved. This saves object program space.
- Alphabetized Cross-Reference Listing (SXREF) -- for easier reference to user-specified names in a program. SXREF performs up to 25 times faster than previous Version 2 source-ordered cross-reference (XREF). Version 3 XREF performance is improved by at least the same amount. The larger the source program, the more that performance is improved. Total compilation time is up to 2 times faster.

- Debugging Facilities that are more powerful and flexible
 - (1) Symbolic Debug Feature -- which provides a symbolic formatted dump at abnormal termination, or a dynamic dump during program execution.
 - (2) Flow Trace Option -- a formatted trace can be requested for a variable number of procedures executed before abnormal termination.
 - (3) Statement Number Option -- provides information about the COBOL statement being executed at abnormal termination.
 - (4) Expanded CLIST and SYM -- for more detailed information about the Data Division and Procedure Division.
 - (5) Relocation Factor -- can be requested to be included in addresses on the object code listing for easier debugging.
 - (6) Working-Storage Location and Size -- When CLIST and SYM are in effect, the starting address and size of Working-Storage are printed.
- System/370 Device Support -- the following devices can be specified:
 - 3211 -- 150-character printer
 - 2319, 3330 -- mass storage (direct access) facilities
 - 3410, 3420 -- tape utility devices
 - 3505, 3525 -- advanced unit-record devices
- ASCII Support -- allows creation and retrieval of tape files written in the American National Standard Code for Information Interchange (ASCII).
- Separately Signed Numeric Data Type -- for more flexible numeric data description. The sign can be a separate character or an overpunch, and can be leading or trailing.

The DOS Full American National Standard COBOL Compiler and Library are packaged as two separate Program Products. The Compiler is Program Product Number 5736-CB2; the Library is Program Product Number 5736-LM2.



In 1959, a group of computer professionals, representing the U.S. Government, manufacturers, universities, and users, formed the Conference On Data Systems Language (CODASYL). At the first meeting, the conference agreed upon the development of a common language for the programming of commercial problems. The proposed language would be capable of continuous change and development, it would be problem-oriented and machine-independent, and it would use a syntax closely resembling English, avoiding the use of special symbols as much as possible. The Common Business Oriented Language (COBOL) which resulted met most of these requirements.

As its name implies, COBOL is especially efficient in the processing of business problems. Such problems involve relatively little algebraic or logical processing; instead, they usually manipulate large files of similar records in a relatively simple way. This means that COBOL emphasizes the description and handling of data items and input/output records.

In the years since 1959, COBOL has undergone considerable refinement and standardization, and a standard COBOL has been approved by ANSI (American National Standards Institute), an industry-wide association of computer manufacturers and users; this standard is called American National Standard COBOL, X3.23-1968.

This publication explains IBM Full American National Standard COBOL, which is compatible with the highest level of American National Standard COBOL and includes a number of IBM extensions to it as well. The compiler supports the processing modules defined in the standard. These processing modules include:

NUCLEUS -- which defines the permissible character set and the basic elements of the language contained in each of the four COBOL divisions: Identification Division, Environment Division, Data Division, and Procedure Division.

TABLE HANDLING -- which allows the definition of tables and making reference to them through subscripts and indexes. A convenient method for searching a table is provided.

SEQUENTIAL ACCESS -- which allows the records of a file to be read or written in a serial manner. The order of reference is implicitly determined by the position of the logical record in the file.

RANDOM ACCESS -- which allows the records of a file to be read or written in a manner specified by the programmer. Specifically defined keys, supplied by the programmer, control successive references to the file.

SORT -- which provides the capability of sorting files in ascending and/or descending order. This feature also includes procedures for handling such files both before and after they have been sorted.

REPORT WRITER -- which allows the programmer to describe the format of a report in the DATA DIVISION, thereby minimizing the amount of PROCEDURE DIVISION coding necessary.

SEGMENTATION -- which allows large problem programs to be split into segments that can then be designated as permanent or overlayable core storage. This assures more efficient use of core storage at object time.

LIBRARY -- which supports the retrieval and updating of pre-written source program entries from a user's library, for inclusion in a COBOL program at compile time. The effect of the compilation of library text is as though the text were actually written as part of the source program.

In this publication, the features included in the NUCLEUS, SEQUENTIAL ACCESS, and RANDOM ACCESS modules are presented as part of the discussion of "Language Considerations" and of the four divisions of a COBOL program. The other five modules -- TABLE HANDLING, SORT, REPORT WRITER, LIBRARY, and SEGMENTATION -- are presented as separate features of American National Standard COBOL.

This manual describes all versions of IBM System/360 Disk Operating System Full American National Standard COBOL. All information relating to the Program Product Version 3 compiler is presented within separate paragraphs. Such paragraphs begin with the heading "Program Product Information -- Version 3," and all following paragraphs pertaining to such information are indented. All information relating to the DOS/VS COBOL Compiler and Library Program Product is included in the separate chapter, ~~"DOS/VS COBOL Considerations"~~.

This chapter gives the reader a general understanding of the principles of IBM Full American National Standard COBOL (hereinafter simply termed "COBOL"). It introduces the reader to COBOL and demonstrates some of the ways in which the language can be used in the solution of commercial problems. This discussion does not define the rules for using COBOL, but rather attempts to explain the basic concepts of the language through relatively simple examples.

The reader who has an understanding of the principles of currently implemented versions of COBOL may wish to go directly to "Language Considerations." Other readers will find many concepts discussed in this chapter of help in using the detailed instructions throughout the rest of this manual.

PRINCIPLES OF COBOL

COBOL is one of a group of high-level computer languages. Such languages are problem oriented and relatively machine independent, freeing the programmer from many of the machine oriented restrictions of assembler language, and allowing him to concentrate instead upon the logical aspects of his problem.

COBOL looks and reads much like ordinary business English. The programmer can use English words and conventional arithmetic symbols to direct and control the complicated operations of the computer. The following are typical COBOL sentences:

```
ADD DIVIDENDS TO INCOME.  
MULTIPLY UNIT-PRICE BY STOCK-ON-HAND  
    GIVING STOCK-VALUE.  
IF STOCK-ON-HAND IS LESS THAN ORDER-POINT  
    MOVE ITEM-CODE TO REORDER-CODE.
```

Such COBOL sentences are easily understandable, but they must be translated into machine language -- the internal instruction codes -- before they can actually be used.

A special systems program, known as a compiler, is first entered into the computer. The COBOL program (referred to as the source program) is then entered into the machine, where the compiler reads it and analyzes it. The COBOL language contains a basic set of reserved words and symbols. Each combination of reserved words and symbols is transformed by the compiler into a definite set of usable machine instructions. in

effect, the programmer has at his disposal a whole series of "prefabricated" portions of the machine-language program he wishes the compiler to construct.

When he writes a COBOL program, he is actually directing the compiler to bring together, in the proper sequence, the groups of machine instructions necessary to accomplish the desired result. From the programmer's instructions, the compiler creates a new program in machine language. This program is known as an object program.

Once the object program has been produced, it may be used at once, or it may be recorded on some external medium and stored for future use. When it is needed, it can then be called upon again and again to process data.

Every COBOL program is processed first when the compiler translates the COBOL program into machine language (compile time), then when the machine language program actually processes the data (execution time).

A simple example illustrates the basic principles of translating a COBOL sentence. To increase the value of an item named INCOME by the value of an item named DIVIDENDS, the COBOL programmer writes the following sentence:

```
ADD DIVIDENDS TO INCOME.
```

Before the compiler can interpret this sentence, it must be given certain information. The programmer describes the data represented by the names DIVIDENDS and INCOME in such a way that the compiler can recognize it, obtain it when needed, and treat it in accordance with its special characteristics.

First, the compiler examines the word ADD. It determines whether or not ADD is one of the COBOL reserved words, that is, words that have clearly defined meanings in COBOL (rather than a word like DIVIDENDS, which is defined by the programmer). ADD is a special kind of reserved word--a COBOL key word. Therefore, the compiler generates the machine instructions necessary to perform an addition and inserts them into the object program.

The compiler next examines the word DIVIDENDS. Because the programmer has supplied data information about DIVIDENDS, the compiler knows where and how DIVIDENDS information is to be placed in core storage, and it inserts into the object program the instructions needed in order to locate and obtain the data.

When the compiler encounters the word TO, it again determines whether or not this is a COBOL reserved word. It is such a word, and the compiler interprets it to mean that the value represented by the name following the word TO, in this case INCOME, must be increased as a result of the addition.

The compiler next examines the word INCOME. Again, it has access to data information about the word. As a result, it is able to place in the object program the instructions necessary to locate and use INCOME data.

The programmer placed a period after the word INCOME. The effect of the period on the COBOL compiler is similar to its effect in the English language. The period tells the compiler that it has reached the last word to which the verb ADD applies, the end of the sentence.

The logical steps we have described are performed by the compiler in creating the object program, although they might not be performed in exactly this sequence. All these preparatory steps are required only in creating the object program. Once created, the object program is used for the actual processing and may be saved for future reference. The source program is not required further, unless the programmer makes a

change in it; in that case, it must be compiled again to create a new object program.

When the machine-language instruction for ADD is actually performed at execution time, the instruction is executed in either of two ways, depending on the format of the data:

1. It directly adds the value of DIVIDENDS to the value of the data representing INCOME, thus giving the new value of INCOME.

or

2. It moves the data representing INCOME into a special work area, or register; then DIVIDENDS is added to it to create the sum, after which the new value of INCOME is returned to the proper area in storage.

In this simple example, the object program could add the two specified items with very few machine instructions. In actual practice, however, some complex COBOL sentences produce dozens of machine instructions. Then, too, a computer can be instructed to repeat a procedure any number of times. A few COBOL sentences can start the computer on operations that could process millions of data records rapidly and accurately.

A SAMPLE COBOL PROGRAM

COBOL is based on English; it uses English words and certain syntax rules derived from English. However, because it is a computer language, it is much more precise than English. The programmer must, therefore, learn the rules that govern COBOL and follow them exactly. These rules are detailed later, beginning in the next chapter. The rest of this chapter gives a general picture of how a COBOL program is put together.

The basic unit of COBOL is the word -- which may be a COBOL reserved word or a programmer-defined word. Reserved words have a specific syntactical meaning to the COBOL compiler, and must be spelled exactly as shown in the reserved word list (see Appendix C). Programmer-defined words are assigned by the user to such items as data-names and procedure-names; they must conform to the COBOL rules for the formation of names.

Reserved words and programmer-defined words are combined by the programmer into clauses (in the Environment and Data Divisions) and statements (in the Procedure Division); clauses and statements must be formed following the specific syntactical rules of COBOL. A clause or a statement specifies only one action to be performed, one condition to be analyzed, or one description of data. Clauses and statements can be combined into sentences. Sentences may be simple (one statement or one clause), or they may be compound (a combination of statements or a combination of clauses). Sentences can be combined into paragraphs, which are named units of logically related sentences, and paragraphs can be further combined into named sections. Both paragraphs and sections can be referred to as procedures, and their names can be referred to as procedure names. Procedures (sections and paragraphs) are combined into divisions.

There are four divisions in each COBOL program. Each is placed in its logical sequence, each has its necessary logical function in the program, and each uses information developed in the divisions preceding it. The four divisions and their sequence are:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

To illustrate how a COBOL program is written, let us create a simplified procedure to record changes in the stocks of office furniture offered for sale by a manufacturer. We will need such data items as an item code to identify each type of product, an item name corresponding to the code, the unit price of each item of stock, the reorder point at which the manufacturer replaces each item, and the amount of stock on hand plus its value for each item. Our procedure will update a MASTER-FILE of all stocks the manufacturer carries by reading a DETAIL-FILE of current transactions, performing the necessary calculations, and placing the updated values in the MASTER-FILE. We will also create an ACTION-FILE of items to be reordered. The MASTER-FILE resides on a direct access (mass storage) disk device; the DETAIL-FILE and ACTION-FILE reside on tape devices.

Many of the examples used in the following discussion have been simplified for greater clarity. Figure 4, at the end of this chapter, shows how the entire UPDATING program would actually be written.

Identification Division

First we must assign a name to our program, presenting the information like this:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. UPDATING.
```

PROGRAM-ID informs the compiler that we have chosen the unique name UPDATING for the program we have written.

In addition to the name of the program, the Identification Division allows us to list the name of the programmer, the date the program was written, and other information that will serve to document the program.

Environment Division

Although COBOL is, to a large degree, machine independent, there are some aspects of any program that depend on the particular computer being used and on its associated input/output devices. In the Environment Division, the characteristics of the computer used may be identified. The location of each file referenced in the program, and how each one of them will be used, must be described.

First we will describe the source computer (the one the compiler uses) and the object computer (the one the object program uses) as follows:

```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-360-F50.  
OBJECT-COMPUTER. IBM-360-F50.
```

This tells us that both computers will be an IBM System/360 model F50.

Next we must identify the files to be used in our program, and assign them to specific input/output devices. This is done in the Input-Output Section.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

```
SELECT MASTER-FILE, ASSIGN TO ...
    ACCESS MODE IS RANDOM
    ACTUAL KEY IS FILEKEY.
SELECT DETAIL-FILE, ASSIGN TO ...
    ACCESS MODE IS SEQUENTIAL.
SELECT ACTION-FILE, ASSIGN TO ...
.
.
.
```

The ellipses (...) in the three foregoing ASSIGN clauses indicate the omission of system-name, an item too complex to illustrate here. System-name is in a special format, and it tells the compiler on which symbolic unit the file will be found, on what kind of device the file resides, and in what way the data is organized within the file.

Our MASTER-FILE resides on a disk pack, which is a mass storage device. Access for these devices can be either RANDOM or SEQUENTIAL. If ACCESS MODE IS RANDOM, then each record within the file can be located directly through the use of a key (identified in the statement ACTUAL KEY IS FILEKEY). For our program we have named this key FILEKEY, and later in the Data Division we will describe it fully. During the processing of our object program, each record will be made available to the user in the sequence that the keys are presented to the system.

Our DETAIL-FILE and our ACTION-FILE reside on tape. This means that ACCESS MODE must be sequential. On tape it is necessary to refer to each successive record in the file in order to find any individual record we might wish to access. Since the compiler assumes that the ACCESS MODE is sequential unless specified otherwise, the ACCESS MODE clause is never needed in describing a tape file.

Data Division

The Data Division of the COBOL program gives a detailed description of all the data to be used in the program -- whether to be read into the machine, used in intermediate processing, or written as output. To simplify this discussion, we will describe only the two most important aspects of data description.

1. We will inform the compiler that we intend to work with one kind of input record, our detail record; one kind of update record, our master record; and one kind of output record, our action record.
2. We will assign data-names to each of the items of data to be used.

First, we must organize the two input records -- a MASTER-RECORD and a DETAIL-RECORD. The MASTER-RECORD will be derived from ledger records that look like those shown in Figure 1.

Item Code	Item Name	Stock on Hand	Unit Price (\$)	Stock Value (\$)	Order Point
A10	2-drawer file cabinets	100	50	5,000	50
A11	3-drawer file cabinets	175	80	14,000	80
A12	4-drawer file cabinets	200	110	22,000	150
B10	Secretarial desks	150	200	30,000	120
B11	Salesmen's desks	50	175	8,750	50
B12	Executive desks	75	500	37,500	60
C10	Secretarial posture chairs	125	50	6,250	140
C11	Side chairs	50	40	2,000	60
C12	Executive swivel chairs	25	150	3,750	20

Figure 1. Typical Ledger Records Used for MASTER-RECORD

There will be a MASTER-RECORD for each item in this list. In defining the data for the compiler, we will make sure that each record is in the same format as all the others. Thus, if we specify the characteristics of a single record, we will have specified the characteristics of the whole set. In this way, all of the master records can be organized into a data set, or file, that we will name MASTER-FILE. Each complete record within the file we will name the MASTER-RECORD, with the individual items of data grouped within it. Accordingly, we will begin our Data Division as follows:

```

DATA DIVISION.
FILE SECTION.
FD MASTER-FILE DATA RECORD IS MASTER-RECORD...
.
.
.
01 MASTER-RECORD.
02 ITEM-CODE...
02 ITEM-NAME...
02 STOCK-ON-HAND...
02 UNIT-PRICE...
02 STOCK-VALUE...
02 ORDER-POINT...
.
.
.

```

The FILE SECTION entry informs the COBOL compiler that the items that follow will describe the format of each file and of each record within each file to be used in the program. The level indicator FD (File Description) introduces the MASTER-FILE itself, and tells the compiler that each entry within MASTER-FILE will be referred to as MASTER-RECORD. The entry with level number 01 identifies the MASTER-RECORD itself, and the subordinate entries with level number 02 describe the subdivisions within the complete MASTER-RECORD. The concept of levels is a basic attribute of COBOL. The highest level is the FD, the next highest level is 01. Level numbers from 02 through 49 may subdivide the record, and the subdivisions themselves can be further subdivided if need be. The smaller the subdivision, the larger the level number must be.

Each of the data items would actually be described more fully than is shown here. In an actual program, for example, we would inform the compiler that each of the items identified as STOCK-ON-HAND, UNIT-PRICE, STOCK-VALUE, and ORDER-POINT would represent positive numeric values of a specific size in a specific form, and so forth. At this point, we need not concern ourselves with these details.

The MASTER-FILE is the main record of current inventory. Changes to this record are made by entering the details of individual transactions or groups of transactions. Thus, receipts of new stocks and shipments to customers will change both STOCK-ON-HAND and STOCK-VALUE. These changes are summarized in the detail record for each item. A typical record would appear in a ledger as shown in Figure 2.

Item Code	Item Name	Receipts	Shipments
B11	Salesmen's desks	25	15

Figure 2. Typical DETAIL-RECORD

We will therefore organize a DETAIL-FILE, made up of individual items to be referred to as DETAIL-RECORD. DETAIL-FILE will be arranged by ITEM-CODE in ascending numerical order.

```

FD  DETAIL-FILE DATA RECORD IS DETAIL-RECORD...
01  DETAIL-RECORD.
    02  ITEM-CODE...
    02  ITEM-NAME...
    02  RECEIPTS...
    02  SHIPMENTS...

```

The ACTION-FILE will contain a list of items to be reordered, plus relevant data:

```

FD  ACTION-FILE DATA RECORD IS ACTION-RECORD...
01  ACTION-RECORD.
    02  ITEM-CODE...
    02  ITEM-NAME...
    02  STOCK-ON-HAND...
    02  UNIT-PRICE...
    02  ORDER-POINT...

```

This completes the description of the files we will use.

Note that the names of data items contained within the files are in many cases identical. Yet each name within each file must be unique, or ambiguities in references to them will occur. Since identical names are used in our data descriptions, we must use a special means of distinguishing between them. The COBOL naming system, with its concept of levels, allows us to make this distinction by reference to some larger group of data of which the item is a part. Thus, ITEM-CODE OF MASTER-RECORD, and ITEM-CODE OF DETAIL-RECORD, and ITEM-CODE OF ACTION-RECORD can be clearly differentiated from each other. The use of a higher level name in this way is called qualification. Qualification is required in making distinctions between otherwise identical names.

Now we must construct the Working-Storage Section of our Data Division. This section describes records and data items that are not part of the files, but are used during the processing of the object program.

For our program, we will need several entries in our Working-Storage Section. Among them will be several items constructed with level numbers, similar to those used to describe the file records.


```

WORKING-STORAGE SECTION.
.
.
.
77  QUOTIENT...
.
.
.
01  THE-KEY...
    02  FILLER...
    02  FILEKEY...
01  ERROR-MESSAGE.
    02  ERROR-MESSAGE-1...
    02  ERROR-MESSAGE-2...
    02  ERROR-MESSAGE-3...

```

We will use THE-KEY record in constructing the FILEKEY. The ERROR-MESSAGE record we will use to create warning messages when errors are encountered during object time processing. The data item named QUOTIENT we have assigned the level number 77. This level number informs the compiler that QUOTIENT is a noncontiguous data item -- that is, that this item has no relationship to any other data item described in the Working-Storage Section. Note that the data items related to each other must be listed after all the noncontiguous data items.

Procedure Division

The Procedure Division contains the instructions needed to solve our problem. To accomplish this, we will use several types of COBOL statements. In constructing our sample program, we will discover how each type of statement can be used to obtain the results we want.

Beginning the Program -- Input Operations

Our first step in building the Procedure Division is to make the records contained in the MASTER-FILE and the DETAIL-FILE available for processing. If we write the statements:

```

PROCEDURE DIVISION.
.
.
.
    OPEN INPUT DETAIL-FILE.
    OPEN I-O MASTER-FILE.

```

the system establishes a line of communication with each file, checks to make sure that each is available for use, brings the first record of the DETAIL-FILE file into special areas of internal storage known as buffers, and does other housekeeping.

The files can now be accessed. Our next statements will therefore be:

```
READ DETAIL-FILE AT END GO TO END-ROUTINE.  
.  
.  
READ MASTER-FILE INVALID KEY PERFORM INPUT-ERROR  
GO TO ERROR-ROUTINE-1.
```

At this point in our program, these two statements make available for processing the first record from each file. (Note that the AT END phrase and the INVALID KEY phrase are necessary in these sentences. Their use will be explained later.) We are now able to begin arithmetic operations upon the data.

Arithmetic Statements

We have already seen that the COBOL language contains the verb ADD. Using this verb, we can add RECEIPTS to STOCK-ON-HAND by writing the COBOL statement:

```
ADD RECEIPTS TO STOCK-ON-HAND.
```

This instructs the program to find the value of RECEIPTS in the DETAIL-RECORD and add it to the value of STOCK-ON-HAND in the MASTER-RECORD. (For the sake of brevity, this example and the ones following have been simplified by omitting the name qualification which would be necessary in actual coding. Figure 4, at the end of this chapter, shows the actual coding necessary.)

Next we must reduce the new value of STOCK-ON-HAND by the amount of SHIPMENTS. The COBOL verb SUBTRACT will accomplish this result for us, and so we write:

```
SUBTRACT SHIPMENTS FROM STOCK-ON-HAND.
```

These two statements, carried out in succession, will produce a current value for STOCK-ON-HAND.

Actually, there is a more concise way to perform this particular calculation. We have broken it into two steps, but COBOL provides another verb which allows us to specify more than one arithmetic operation in a single statement. This is the verb COMPUTE.

```
COMPUTE STOCK-ON-HAND = STOCK-ON-HAND + RECEIPTS - SHIPMENTS.
```

A COMPUTE statement is always interpreted to mean that the value on the left of the equal sign will be changed to equal the value resulting from the calculation specified on the right. The calculation on the right of the equal sign is evaluated from left to right. That is, in our example, the addition is performed first and then the subtraction.

The name STOCK-ON-HAND occurs twice in this sentence, but this causes no difficulty. The expression to the right is calculated first; thus, it is the current value of STOCK-ON-HAND that is used as the basis for computing the new value. When this new value has been calculated, it replaces the old value of STOCK-ON-HAND in the MASTER-RECORD.

So far we have brought only the value of STOCK-ON-HAND up to date, but a change in this value will also cause a change in STOCK-VALUE. We will assume that this figure does not include allowances for quantity discounts, damage to stock, or other such factors, and that STOCK-VALUE is nothing more than the unit price multiplied by the number of items currently in stock. COBOL provides us with a MULTIPLY verb, which permits us to accomplish this:

```
MULTIPLY STOCK-ON-HAND BY UNIT-PRICE GIVING STOCK-VALUE.
```

The result of the multiplication will be placed in the MASTER-RECORD as the new value of STOCK-VALUE. Within the program, this statement must be executed after the COMPUTE statement we wrote earlier, since STOCK-ON-HAND must be the updated, not the original, value.

Conditional Statements

There are instructions in COBOL that examine data to determine whether or not some condition is present and, depending on what is found, to carry out an appropriate course of action.

The MASTER-RECORD contains an item called ORDER-POINT. An item is to be reordered when its stock has been reduced either to or below its order point. Let us assume that we have written a procedure for initiating such an order, and that we have given the name REORDER-ROUTINE to this procedure. We then write the following two sentences:

```
IF STOCK-ON-HAND IS LESS THAN ORDER-POINT
  PERFORM REORDER-1...
IF STOCK-ON-HAND IS EQUAL TO ORDER-POINT
  PERFORM REORDER-1...
```

in order to compare the present value of STOCK-ON-HAND with the value of ORDER-POINT. If STOCK-ON-HAND is a smaller value, the COBOL verb PERFORM causes a transfer of control to the paragraph named REORDER-1. If STOCK-ON-HAND is not less than ORDER-POINT, our next instruction is evaluated. If the values are equal, control is transferred to REORDER-1. If the values are not equal, control is transferred to the next instruction.

It is permissible, in COBOL, to combine the two tests into one:

```
IF STOCK-ON-HAND IS LESS THAN ORDER-POINT OR EQUAL TO
  ORDER-POINT PERFORM REORDER-1...
```

Here we are writing a compound condition with an implied subject. STOCK-ON-HAND, the subject of the first condition, is understood to be the subject of the second condition as well. Compound conditions increase the flexibility of COBOL and make the handling of many kinds of problems easier.

In this example, we tested successively for two conditions out of three. Unless the programmer has some need to distinguish between these two conditions (and he might), it would be simpler to test for the third condition instead:

```
IF STOCK-ON-HAND IS GREATER THAN ORDER-POINT NEXT SENTENCE
  ELSE PERFORM REORDER-1...
```

The words NEXT SENTENCE have a special meaning in COBOL. When IF STOCK-ON-HAND IS GREATER THAN ORDER-POINT is true, NEXT SENTENCE takes effect. Every instruction in the balance of the IF sentence is ignored, and control is transferred to the sentence following.

The test can be simplified even further, since COBOL allows us to express negation:

```
IF STOCK-ON-HAND IS NOT GREATER THAN ORDER-POINT
  PERFORM REORDER-1...
```

If the value of STOCK-VALUE is less than or equal to that of ORDER-POINT, control is transferred to REORDER-1. If the value is greater, control automatically passes to the next successive sentence.

The actual rules for specifying tests and comparisons will be given in a subsequent chapter.

Handling Possible Errors

Let us write one more conditional statement:

```
IF STOCK-ON-HAND IS LESS THAN ZERO...
  GO TO ERROR-WRITE.
```

One would expect that the smallest value STOCK-ON-HAND could assume would be zero. If a negative record were processed, the values found would probably be completely erroneous. To prevent this, the programmer could anticipate the possibility of error and write a special routine to be executed whenever the value of STOCK-ON-HAND was found to be negative. Such a routine could stop the processing of this record, print out the erroneous data, and proceed automatically to process the records following. The more comprehensive a programmer makes his error checking, the less likely it is that inaccurate information will pass through without being marked for special attention.

Data-Manipulation Statements

We saw in the foregoing that if the value of STOCK-ON-HAND fell below a certain point, control would be passed to a special sequence of instructions named REORDER-1. Our output ACTION-FILE has been set up for just this purpose. The bulk of REORDER-1 could consist of data-manipulation statements; that is, instructions which move the necessary data items from the MASTER-RECORD area in storage to that area reserved for the ACTION-FILE records. The COBOL verb MOVE can be used to accomplish this. We must explain here that the verb MOVE does not mean an actual physical movement of data. Instead, it means that the data items from MASTER-RECORD are copied into ACTION-RECORD. Items within MASTER-RECORD are not destroyed when a MOVE statement is executed, and are available for further processing. Individual items

contained in ACTION-RECORD before the operation, however, are replaced when the statement is executed. Our MOVE statements will be written:

```
MOVE ITEM-CODE OF MASTER-RECORD TO ITEM-CODE
      OF ACTION-RECORD.
MOVE ITEM-NAME OF MASTER-RECORD TO ITEM-NAME
      OF ACTION-RECORD.
MOVE STOCK-ON-HAND OF MASTER-RECORD TO
      STOCK-ON-HAND OF ACTION-RECORD.
MOVE UNIT-PRICE OF MASTER-RECORD TO UNIT-PRICE
      OF ACTION-RECORD.
MOVE ORDER-POINT OF MASTER-RECORD TO ORDER-POINT
      OF ACTION-RECORD.
```

With these five statements, we have set up the ACTION-RECORD to be written in the ACTION-FILE. However, there is another and easier method for the programmer to specify the five MOVE operations by taking advantage of the qualification system in naming:

```
MOVE CORRESPONDING MASTER-RECORD TO ACTION-RECORD.
```

The word CORRESPONDING indicates that those data items with names which are identical in both records are to be copied from MASTER-RECORD into ACTION-RECORD. Thus, five MOVE statements are replaced by one.

Output Operations

When all arithmetic and data-manipulation statements have been executed, we will write the results in some form. COBOL allows us to do this with a WRITE instruction.

```
WRITE MASTER-RECORD INVALID KEY ...
      GO TO ERROR-WRITE.
```

Or, if we were to indicate that an item was to be reordered, we could write the following:

```
WRITE ACTION-RECORD.
```

In either case, the record would be recorded on the output device specified for the file in the Environment Division; its format would be determined by the Data Division description of the file.

Procedure Branching Statements

In our inventory problem, there will be as many master records as there are kinds of furniture in stock, and there will be a varying number of detail records. We must read each successive DETAIL-RECORD in DETAIL-FILE, until every one of the records in the file has been processed.

Each time a DETAIL-RECORD is read, we will perform calculations upon its ITEM-CODE in order to produce our FILEKEY. FILEKEY will then be used to find a matching record in MASTER-RECORD. If a matching record cannot be found, either the DETAIL-RECORD is in error, or the MASTER-RECORD is missing from the file and we must mark that record for special processing. Consider the series of statements in Figure 3.

You will note that several new elements have been added to the arithmetic statements and conditional phrases we have already discussed. First, there are the elements that extend to the left of the other statements. These elements are the procedure-names we described earlier. Each procedure-name indicates the beginning of a paragraph or a section within the program, and each indicates a reference point for programmer-specified transfer of control. When a procedure is entered, each logically successive instruction is processed in turn.

The procedure-names give us a means of controlling the processing of successive items in our DETAIL-FILE. If, for example, we have finished processing one complete DETAIL-RECORD and wish to begin processing the next, control must be transferred to NEXT-DETAIL-RECORD-ROUTINE. This is accomplished through the use of the COBOL verb GO TO, which transfers control to the procedure indicated, as in the statement:

```
GO TO NEXT-DETAIL-RECORD-ROUTINE.
```

Processing then continues with the first sentence following the procedure name NEXT-DETAIL-RECORD-ROUTINE. Note the many other examples of the GO TO statement in our program. Each gives us the means of transferring control from one procedure to another.

Another way in which to control the processing of a series of records is through the use of the COBOL verb PERFORM. Like the verb GO TO, the verb PERFORM specifies a transfer to the first sentence of a routine. In addition, PERFORM provides various ways of determining the manner in which the procedure is to be processed.

Within the COMPUTATION-ROUTINE, there is a statement which uses the COBOL verb PERFORM:

```
IF STOCK-ON-HAND IN MASTER-RECORD IS LESS THAN ZERO  
    PERFORM DATA-ERROR GO TO ERROR-WRITE.
```

When STOCK-ON-HAND is computed to be less than zero, an error condition has occurred. First, the compiler is instructed to transfer control to a procedure named DATA-ERROR. Within DATA-ERROR, there is a MOVE statement which copies the characters within quotation marks ("DATA ERROR ON INPUT ") into the area of storage reserved for ERROR-MESSAGE-1. (The characters within quotation marks are what is known as a literal -- because they literally mean themselves. When ERROR-MESSAGE is displayed, these words will be an actual part of the error message.) Control is now transferred back to the next statement following the PERFORM statement, which is the GO TO ERROR-WRITE statement.

```

NEXT-DETAIL-RECORD-ROUTINE.
  READ DETAIL-FILE AT END GO TO END-ROUTINE-1.
  .
  .
  READ MASTER-FILE INVALID KEY PERFORM INPUT-ERROR
  GO TO ERROR-WRITE.
COMPUTATION-ROUTINE.
  .
  .
  IF STOCK-ON-HAND IN MASTER-RECORD IS LESS THAN ZERO
  PERFORM DATA-ERROR GO TO ERROR-WRITE.
  .
  .
  IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
  ORDER-POINT IN MASTER-RECORD PERFORM REORDER-1
  THRU REORDER-2.
WRITE-MASTER-ROUTINE.
  .
  .
  GO TO NEXT-DETAIL-RECORD-ROUTINE.
REORDER-1.
  GO TO SWITCH-ROUTINE.
SWITCH-ROUTINE.
  ALTER REORDER-1 TO REORDER-2
  END-ROUTINE-1 TO END-ROUTINE-3.
  OPEN OUTPUT ACTION-FILE.
REORDER-2.
  MOVE CORRESPONDING MASTER-RECORD TO ACTION-RECORD.
  WRITE ACTION-RECORD.
ERROR-WRITE.
  .
  .
  GO TO NEXT-DETAIL-RECORD-ROUTINE.
INPUT-ERROR.
  MOVE " KEY ERROR ON INPUT " TO ERROR-MESSAGE-1.
  .
  .
DATA-ERROR.
  MOVE "DATA ERROR ON INPUT " TO ERROR-MESSAGE-1.
  .
  .
END-ROUTINE-1.
  GO TO END-ROUTINE-2.
END-ROUTINE-3.
  CLOSE ACTION-FILE.
END-ROUTINE-2.
  CLOSE DETAIL-FILE.
  CLOSE MASTER-FILE.
  STOP RUN.

```

Figure 3. Illustration of Procedure Branching

Note that within COMPUTATION-ROUTINE there is another PERFORM statement that is processed in a similar manner:

```
IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
ORDER-POINT IN MASTER-RECORD
PERFORM REORDER-1 THRU REORDER-2.
```

This time, the PERFORM statement instructs the object program to process several paragraphs before returning control to the next successive statement. Thus, when this PERFORM statement is executed, control is transferred to REORDER-1. This paragraph is executed, the next paragraph, SWITCH-ROUTINE, is also executed, and then all the statements contained in REORDER-2 are executed, at which point control is returned to the first statement in WRITE-MASTER-ROUTINE -- the next successive statement after the PERFORM statement.

A PERFORM statement may specify that a single section or paragraph be processed, or, if the desired procedure consists of more than one section or paragraph, it can specify two names that identify the beginning and the end of the procedure.

GO TO and PERFORM statements may seem to do much the same job. Yet there are specific reasons that will cause the programmer to choose one over the other. On the one hand, the programmer may wish to transfer control to the same procedure from two entirely different sections of the program. In this case, PERFORM offers the most convenient method of returning to the point from which the transfer was made. On the other hand, if the programmer wishes to proceed to a portion of the program without specifying a return to the current routine, a GO TO statement will provide the best method of making the transfer.

In addition to the GO TO and PERFORM statements, there is another COBOL statement that affects procedure branching: the ALTER statement.

In any given execution of our object program, we may or may not use our ACTION-FILE. Only if some item in STOCK-ON-HAND has fallen below REORDER-POINT will it be necessary to create an ACTION-RECORD. Therefore, depending upon the data that is being processed, we will open ACTION-FILE only if and when such an operation is necessary.

Suppose that for the first time in a particular execution of our object program we have encountered a value for STOCK-ON-HAND that indicates it must be reordered. The statement:

```
IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
ORDER-POINT IN MASTER-RECORD
PERFORM REORDER-1 THRU REORDER-2.
```

instructs the compiler, when STOCK-ON-HAND is not greater than ORDER-POINT, to transfer control to the first sentence in REORDER-1. REORDER-1 consists of but one statement:

```
GO TO SWITCH-ROUTINE.
```

SWITCH-ROUTINE, as it happens, is the next paragraph, and it contains an ALTER statement:

```
ALTER REORDER-1 TO REORDER-2
END-ROUTINE-1 TO END-ROUTINE-3.
```

This statement instructs the compiler to substitute the words REORDER-2 for SWITCH-ROUTINE (within REORDER-1), and END-ROUTINE-3 for END-ROUTINE-2 (within END-ROUTINE-1). Since, at the time the ALTER statement is executed, we are already beyond the point at which the

substitution is to be made in REORDER-1, we continue processing each sequential statement until we reach the end of REORDER-2. We open ACTION-FILE, and so forth, until we return control to the next statement following the PERFORM statement.

However, in this execution of our object program, the next time we must reorder an item, a different sequence of statements is performed. The program transfers control to REORDER-1, but now the GO TO statement within REORDER-1 has a different operand. Instead of SWITCH-ROUTINE, the program is now instructed to transfer control to the paragraph named REORDER-2. Through use of the ALTER statement, we have created a switch that bypasses the OPEN ACTION-FILE statement in subsequent processing of reordered items, since the OPEN statement need be executed but once in any execution of our object program.

Similarly, if ACTION-FILE was never opened in this execution of our object program, it is not necessary to close it. Therefore, the second part of the ALTER statement:

```
END-ROUTINE-1 TO END-ROUTINE-3
```

allows alternate paths of program flow, depending on whether or not this ALTER statement was ever executed. The precise rules for programming the ALTER statement are given later in this publication; note, however, the increased programming flexibility it offers.

Ending the Program

One last step in the logic of our inventory program must now be taken. We have obtained the update information from a record, performed the needed arithmetic calculations, moved the data from one area of storage to another, and written the decision-making and procedure-branching instructions necessary to take care of special cases and to process each succeeding record. Then we have written the updated information into the MASTER-FILE, and, when necessary, have written the ACTION-FILE. We must now terminate the program after all records have been acted upon. Remember that we wrote our first READ statement as follows:

```
READ DETAIL-FILE AT END GO TO END-ROUTINE-1.
```

END-ROUTINE-1 will consist of the few instructions necessary to terminate operations for this program.

Just as the programmer made all the files available to the system with a set of OPEN instructions, he must now disconnect these same files with another series:

```
END-ROUTINE-1.  
  GO TO END-ROUTINE-2.  
END-ROUTINE-3.  
  CLOSE ACTION-FILE.  
END-ROUTINE-2.  
  CLOSE DETAIL-FILE.  
  CLOSE MASTER-FILE.
```

These instructions initiate necessary housekeeping routines. (Note here that, in our program, ACTION-FILE will be closed only if REORDER-1 THRU REORDER-2 has been performed and the ALTER statement has been executed.) Once a file has been closed, it cannot be accessed by the program again. The programmer now writes one last COBOL instruction, and it must be at the logical end of his processing:

STOP RUN.

At this point, COBOL ending procedures are initiated, and the execution of the program is halted.

This is only a general picture of the way in which a COBOL program works. The following chapters in this manual give detailed descriptions of all four divisions within a COBOL program, with explicit instructions for correct programming in IBM Full American National Standard COBOL.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. UPDATING.
REMARKS. THIS IS A SIMPLIFIED UPDATE PROGRAM, USED AS AN
        EXAMPLE OF BASIC COBOL TECHNIQUES. THE PROGRAM IS
        EXPLAINED IN DETAIL IN THE INTRODUCTION TO THIS MANUAL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-F50.
OBJECT-COMPUTER. IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT MASTER-FILE ASSIGN TO SYS015-DA-2311-A-MASTER
                ACCESS MODE IS RANDOM
                ACTUAL KEY IS FILEKEY.
        SELECT DETAIL-FILE ASSIGN TO SYS007-UT-2400-S-INFILE
                ACCESS IS SEQUENTIAL.
        SELECT ACTION-FILE ASSIGN TO SYS008-UT-2400-S-OUTFILE.
DATA DIVISION.
FILE SECTION.
FD MASTER-FILE LABEL RECORDS ARE STANDARD
DATA RECORD IS MASTER-RECORD.
01 MASTER-RECORD.
    02 ITEM-CODE          PICTURE X(3).
    02 ITEM-NAME          PICTURE X(29).
    02 STOCK-ON-HAND      PICTURE S9(6)      USAGE COMP SYNC.
    02 UNIT-PRICE         PICTURE S999V99    USAGE COMP SYNC.
    02 STOCK-VALUE        PICTURE S9(9)V99   USAGE COMP SYNC.
    02 ORDER-POINT        PICTURE S9(3)      USAGE COMP SYNC.
FD DETAIL-FILE LABEL RECORDS ARE OMITTED
DATA RECORD IS DETAIL-RECORD.
01 DETAIL-RECORD.
    02 ITEM-CODE          PICTURE X(3).
    02 ITEM-NAME          PICTURE X(29).
    02 RECEIPTS           PICTURE S9(3)      USAGE COMP SYNC.
    02 SHIPMENTS          PICTURE S9(3)      USAGE COMP SYNC.
FD ACTION-FILE LABEL RECORDS ARE OMITTED
DATA RECORD IS ACTION-RECORD.
01 ACTION-RECORD.
    02 ITEM-CODE          PICTURE X(3).
    02 ITEM-NAME          PICTURE X(29).
    02 STOCK-ON-HAND      PICTURE S9(6)      USAGE COMP SYNC.
    02 UNIT-PRICE         PICTURE S999V99    USAGE COMP SYNC.
    02 ORDER-POINT        PICTURE S9(3)      USAGE COMP SYNC.
WORKING-STORAGE SECTION.
77 SAVE                  PICTURE S9(10)     USAGE COMP SYNC.
77 QUOTIENT              PICTURE S9999      USAGE COMP SYNC.
01 KEY-ACTUAL.
    02 M                  PICTURE S999 COMP SYNC VALUE ZEROS.
    02 BB                 PICTURE S9 COMP SYNC VALUE ZEROS.
    02 CC                 PICTURE S999 COMP SYNC VALUE ZEROS.
    02 HH                 PICTURE S999 COMP SYNC.
    02 R                  PICTURE X VALUE LOW-VALUE.
    02 RECORD-ID          PICTURE X(29).
01 THE-KEY REDEFINES KEY-ACTUAL.
    02 FILLER              PICTURE X.
    02 FILEKEY             PICTURE X(37).
01 TRACK1                PICTURE 9(4).
01 TRACK2 REDEFINES TRACK1 COMP.
    02 CYL                 PICTURE S999.
    02 HEAD                PICTURE S9.
01 ERROR-MESSAGE.
    02 ERROR-MESSAGE-1    PICTURE X(20).
    02 ERROR-MESSAGE-2    PICTURE X(36).
    02 ERROR-MESSAGE-3    PICTURE X(46).

```

Figure 4. Complete UPDATING Program (Part 1 of 2)

```

PROCEDURE DIVISION.
OPEN-FILES-ROUTINE.
    OPEN INPUT DETAIL-FILE.
    OPEN I-O MASTER-FILE.
NEXT-DETAIL-RECORD-ROUTINE.
    READ DETAIL-FILE AT END GO TO END-ROUTINE-1.
NEXT-MASTER-RECORD-ROUTINE.
    MOVE ITEM-CODE IN DETAIL-RECORD TO SAVE.
    DIVIDE 19 INTO SAVE GIVING QUOTIENT
        REMAINDER TRACK1.
    ADD 1020 TO TRACK1.
    MOVE ITEM-NAME IN DETAIL-RECORD TO RECORD-ID.
    MOVE HEAD TO HH. MOVE CYL TO CC.
    READ MASTER-FILE INVALID KEY
        PERFORM INPUT-ERROR GO TO ERROR-WRITE.
COMPUTATION-ROUTINE.
    COMPUTE STOCK-ON-HAND IN MASTER-RECORD = STOCK-ON-HAND
        IN MASTER-RECORD + RECEIPTS - SHIPMENTS.
    IF STOCK-ON-HAND IN MASTER-RECORD IS LESS THAN ZERO
        PERFORM DATA-ERROR GO TO ERROR-WRITE.
    MULTIPLY STOCK-ON-HAND IN MASTER-RECORD BY UNIT-PRICE
        IN MASTER-RECORD GIVING STOCK-VALUE
        IN MASTER-RECORD.
    IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
        ORDER-POINT IN MASTER-RECORD PERFORM REORDER-1
        THRU REORDER-2.
WRITE-MASTER-ROUTINE.
    WRITE MASTER-RECORD INVALID KEY
        PERFORM OUTPUT-ERROR GO TO ERROR-WRITE.
    GO TO NEXT-DETAIL-RECORD-ROUTINE.
REORDER-1. GO TO SWITCH-ROUTINE.
SWITCH-ROUTINE.
    ALTER REORDER-1 TO REORDER-2
        END-ROUTINE-1 TO END-ROUTINE-3.
    DISPLAY "ACTION FILE UTILIZED".
    OPEN OUTPUT ACTION-FILE.
REORDER-2.
    MOVE CORRESPONDING MASTER-RECORD TO ACTION-RECORD.
    WRITE ACTION-RECORD.
ERROR-WRITE.
    MOVE DETAIL-RECORD TO ERROR-MESSAGE-2.
    DISPLAY ERROR-MESSAGE.
    GO TO NEXT-DETAIL-RECORD-ROUTINE.
INPUT-ERROR.
    MOVE " KEY ERROR ON INPUT " TO ERROR-MESSAGE-1.
    MOVE SPACES TO ERROR-MESSAGE-3.
DATA-ERROR.
    MOVE "DATA ERROR ON INPUT " TO ERROR-MESSAGE-1.
    MOVE MASTER-RECORD TO ERROR-MESSAGE-3.
OUTPUT-ERROR.
    MOVE "KEY ERROR ON OUTPUT " TO ERROR-MESSAGE-1.
    MOVE SPACES TO ERROR-MESSAGE-3.
END-ROUTINE-1.
    GO TO END-ROUTINE-2.
END-ROUTINE-3.
    CLOSE ACTION-FILE.
END-ROUTINE-2.
    CLOSE DETAIL-FILE.
    CLOSE MASTER-FILE.
    STOP RUN.

```

Figure 4. Complete UPDATING Program (Part 2 of 2)

PART I -- LANGUAGE CONSIDERATIONS

- STRUCTURE OF THE LANGUAGE
- ORGANIZATION OF THE COBOL PROGRAM
- METHODS OF DATA REFERENCE
- USE OF THE COBOL CODING FORM
- FORMAT NOTATION

STRUCTURE OF THE LANGUAGE

The COBOL language is so structured that the programmer can write his individual problem program within a framework of words that have particular meaning to the COBOL compiler. The result is the performance of a standard action on specific units of data. For example, in a COBOL statement such as MOVE NET-SALES TO CURRENT-MONTH, the words MOVE and TO indicate standard actions to the COBOL compiler. NET-SALES and CURRENT-MONTH are programmer-defined words which refer to particular units of data being processed by his problem program.

COBOL CHARACTER SET

The complete character set for COBOL consists of the following 51 characters:

<u>Character</u>	<u>Meaning</u>
0,1,...,9	digit
A,B,...,Z	letter
	space
+	plus sign
-	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
\$	currency sign
,	comma
;	semicolon
.	period (decimal point)
" or '	quotation mark
(left parenthesis
)	right parenthesis
>	"greater than" symbol
<	"less than" symbol

Note: This compiler's default option for the quotation mark is the apostrophe ('). Unless the default option is overridden, the quotation mark (") may not be used. If conformance with the standard character set is desired, the programmer must specify the quotation mark (") through a CBL card at compile time. If the quotation mark is thus specified, the apostrophe (') may not be used.

Note: In addition to these 51 characters, the COBOL compiler will process (e.g., in a VALUE IS clause or in an IF statement) those multiple characters which function as return codes for CICS.

Characters Used in Words

The characters used in words in a COBOL source program are the following:

0 through 9
A through Z
- (hyphen)

A word is composed of a combination of not more than 30 characters chosen from the character set for words. The word cannot begin or end with a hyphen.

Character Set

Characters Used for Punctuation

The following characters are used for punctuation:

<u>Character</u>	<u>Meaning</u>
	space
,	comma
;	semicolon
.	period
" or '	quotation mark
(left parenthesis
)	right parenthesis

The following general rules of punctuation apply in writing a COBOL source program:

1. When any punctuation mark is indicated in a format in this publication, it is required in the program.
2. A period, semicolon, or comma, when used, must not be preceded by a space, but must be followed by a space.
3. A left parenthesis must not be followed immediately by a space; a right parenthesis must not be preceded immediately by a space.
4. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except within nonnumeric literals.
5. An arithmetic operator or an equal sign must always be preceded by a space and followed by a space. A unary operator may be preceded by a left parenthesis.
6. A comma may be used as a separator between successive operands of a statement. An operand of a statement is shown in a format as a lower-case word.
7. A comma or a semicolon may be used to separate a series of clauses. For example, DATA RECORD IS TRANSACTION, RECORD CONTAINS 80 CHARACTERS.
8. A semicolon may be used to separate a series of statements. For example, ADD A TO B; SUBTRACT B FROM C.
9. The word THEN may be used to separate a series of statements. For example, ADD A TO B THEN SUBTRACT B FROM C.

Characters Used for Editing

Editing characters are single characters or specific two-character combinations belonging to the following set:

<u>Character</u>	<u>Meaning</u>
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppression
*	check protection
\$	currency sign
,	comma
.	period (decimal point)

(For applications, see the discussion of alphanumeric edited and numeric edited data items in "Data Division.")

Characters Used in Arithmetic Expressions

The characters used in arithmetic expressions are as follows:

<u>Character</u>	<u>Meaning</u>
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Arithmetic expressions are used in the COMPUTE statement and in relation conditions (see "Procedure Division" for more details).

Characters Used for Relation-conditions

A relation character is a character that belongs to the following set:

<u>Character</u>	<u>Meaning</u>
>	greater than
<	less than
=	equal to

Relation characters are used in relation-conditions (discussed in "Procedure Division").

TYPES OF WORDS

A word is composed of a combination of not more than 30 characters chosen from the character set for words. The word cannot begin or end with a hyphen.

Words

The space (blank) is not an allowable character in a word; the space is a word separator. Wherever a space is used as a word separator, more than one may be used.

A word is terminated by a space, or by a period, right parenthesis, comma, or semicolon.

Reserved Words

Reserved words exist for syntactical purposes and must not appear as user-defined words. However, reserved words may appear as nonnumeric literals, i.e., a reserved word may be enclosed in quotation marks. When used in this manner, they do not take on the meaning of reserved words and violate no syntactical rules.

There are three types of reserved words:

1. Key Words. A key word is a word whose presence is required in a COBOL entry. Such words are upper case and underlined in the formats given in this publication.

Key words are of three types:

- a. Verbs such as ADD, READ, and ENTER.
 - b. Required words, which appear in statement and entry formats, such as the word TO in the ADD statement.
 - c. Words that have a specific functional meaning, such as ZERO, NEGATIVE, SECTION, TALLY, etc.
2. Optional Words. Within each format, upper case words that are not underlined are called optional words because they may appear at the user's option. The presence or absence of each optional word in the source program does not alter the compiler's translation. Misspelling of an optional word, or its replacement by another word of any kind, is not allowed.
 3. Connectives. There are three types of connectives:
 - a. Qualifier connectives, which are used to associate a data-name or paragraph-name with its qualifier. The qualifier connectives are OF and IN (see "Methods of Data Reference").
 - b. Series connectives, which link two or more consecutive operands. The series connective is the comma (,).
 - c. Logical connectives that are used in compound conditions. The logical connectives are AND, OR, AND NOT, and OR NOT (see "Conditions").

Note: Abbreviations (such as PIC for PICTURE) are allowed for some reserved words; the abbreviation is the equivalent of the complete word. For the formats in which they are allowable, such abbreviations are shown in the format. The reserved words THRU and THROUGH are equivalent. In statement formats, wherever the reserved word THRU appears, the word THROUGH is also allowed.

Names

There are three types of names used in a COBOL program:

1. A data-name is a word that contains at least one alphabetic character and identifies a data item in the Data Division. The following are formed according to the rules for data-names:

file-names
 index-names
 mnemonic-names
 record-names
 report-names
 sort-file-names
 sort-record-names

2. A condition-name is a name given to a specific value, set of values, or range of values, within the complete set of values that a particular data item may assume. The data item itself is called a conditional variable. The condition-name must contain at least one alphabetic character (see "Data Division" and the discussion of "Special-Names" in "Environment Division").
3. A procedure-name is either a paragraph-name or a section-name. A procedure-name may be composed solely of numeric characters. Two numeric procedure-names are equivalent if, and only if, they are composed of the same number of digits and have the same value (see "Procedure Division"). The following are formed according to the rules for procedure-names:

library-names
 program-names

Note: The first 8 characters of a file-name must be unique to avoid duplicate names.

Special-Names

Special-names are used in the SPECIAL-NAMES paragraph of the Environment Division. The term special-name refers to a mnemonic-name. A mnemonic-name is a programmer-defined word that is associated in the Environment Division with a function-name: function-names are names with a fixed meaning, defined by IBM.

In the Procedure Division, mnemonic-name can be written in place of its associated function-name in any format where such substitution is valid. The formation of a mnemonic-name follows the rules for formation of a data-name (see "Special-Names" in "Environment Division").

CONSTANTS

A constant is a unit of data whose value is not subject to change. There are two types of constants: literals and figurative constants.

Literals

A literal is a string of characters whose value is determined by the set of characters of which the literal is composed. Every literal belongs to one of two categories, numeric and nonnumeric.

Literals

NUMERIC LITERALS: There are two types of numeric literals: fixed-point and floating-point.

A fixed-point numeric literal is defined as a string of characters chosen from the digits 0 through 9, the plus sign, the minus sign, and the decimal point. The literal -0 is treated by the compiler as a +0. Every fixed-point numeric literal:

1. must contain from 1 through 18 digits.
2. must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.
3. must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere in the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

(See discussion of fixed-point numeric items in "Data Division.")

A floating-point numeric literal is a data item whose potential range of value is too great for fixed-point representation. A floating-point literal must have the form:

[sign]mantissa E[sign]exponent

A floating-point literal must appear as a continuous string of characters with no intervening spaces. The plus or minus signs preceding the mantissa and exponent are the only optional characters within the format. The mantissa consists of from 1 through 16 digits with a required decimal point.

The exponent is represented immediately to the right of the mantissa by the symbol E, followed by a plus or minus sign (if a sign is given) and one or two digits. The magnitude of the number represented by a floating-point literal must not exceed $.72 \times (10^{76})$. A zero exponent must be written as 0 or 00. It is assumed that an unsigned exponent is positive.

The value of the literal is the product of the mantissa and ten raised to the power given by the exponent. For example, the literal

`+.72E+76`

has the value

$.72 \times 10^{76}$

(See discussion of floating-point numeric items in "Data Division.")

If the literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal.

NONNUMERIC LITERALS: A nonnumeric literal is defined as a string of any allowable characters in the Extended Binary Coded Decimal Interchange Code (EBCDIC) set, excluding the quotation mark character. A nonnumeric literal may be composed of from 1 through 120 characters enclosed in quotation marks. Any spaces within the quotation marks are part of the nonnumeric literal and, therefore, are part of the value. All non-numeric literals are in the alphanumeric category.

Figurative Constants

A figurative constant is a constant to which a specific data-name has been assigned. These data-names are reserved words. Such a data-name must not be enclosed in quotation marks when used as a figurative constant. The singular and plural forms of a figurative constant are equivalent and may be used interchangeably.

A figurative constant may be used in place of a literal wherever a literal appears in a format. There is one exception to this rule: if the literal is restricted to numeric characters, only the figurative constant ZERO (ZEROES, ZEROS) is allowed.

The fixed data-names and their meanings are as follows:

<u>ZERO</u> <u>ZEROES</u> <u>ZEROS</u>	Represents the value 0, or one or more occurrences of the character 0, depending on context.
<u>SPACE</u> <u>SPACES</u>	Represents one or more blanks or spaces.
<u>HIGH-VALUE</u> <u>HIGH-VALUES</u>	Represents one or more occurrences of the character that has the highest value in the computer's collating sequence. The character for HIGH-VALUE is the hexadecimal 'FF'.
<u>LOW-VALUE</u> <u>LOW-VALUES</u>	Represents one or more occurrences of the character that has the lowest value in the computer's collating sequence. The character for LOW-VALUE is the hexadecimal '00'.
<u>QUOTE</u> <u>QUOTES</u>	Represents one or more occurrences of the quotation mark character. The word QUOTE (QUOTES) cannot be used in place of a quotation mark to enclose a nonnumeric literal.
<u>ALL</u> literal	Represents one or more occurrences of the string of characters composing the literal. The literal must be either a nonnumeric literal or a figurative constant other than the ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.

Special Registers

SPECIAL REGISTERS

The compiler generates storage areas that are primarily used to store information produced with the use of special COBOL features; these storage areas are called special registers.

TALLY

The word TALLY is the name of a special register whose implicit description is that of an integer of five digits without an operational sign, and whose implicit USAGE is COMPUTATIONAL. The primary use of the TALLY register is to hold information produced by the EXAMINE statement. References to TALLY may appear wherever an elementary data item of integral value may appear (see the "EXAMINE Statement" in "Procedure Division").

LINE-COUNTER

LINE-COUNTER is a numeric counter that is generated by the Report Writer. (For a complete discussion, see "Report Writer.")

PAGE-COUNTER

PAGE-COUNTER is a numeric counter that is generated by the Report Writer. (For a complete discussion, see "Report Writer.")

CURRENT-DATE

CURRENT-DATE is an 8-byte alphanumeric field, valid only as the sending field in a MOVE statement. The format of these eight bytes is MM/DD/YY (month/day/year) or DD/MM/YY (day/month/year).

TIME-OF-DAY

TIME-OF-DAY is a 6-byte external-decimal field, valid only as the sending field in a MOVE statement. The format is HHMMSS (hour, minute, second).

COM-REG

COM-REG is an 11-byte alphanumeric field. This field corresponds to bytes 12 through 22 of the DOS Communication Region. COM-REG is valid only as the sending or receiving field in a MOVE statement. When COM-REG is used as the receiving field in a MOVE statement, the sending field must be 11 bytes in length.

(The use of CURRENT-DATE, TIME-OF-DAY, and COM-REG is explained in the Programmer's Guides (as cited in "Preface").)

SORT-RETURN

SORT-FILE-SIZE

SORT-CORE-SIZE

SORT-MODE-SIZE

The foregoing four registers are used by the Sort feature and are described under "Sort."

NSTD-REELS

NSTD-REELS is a binary field whose PICTURE is S999 and whose maximum value is 255. It is used to indicate the number of reels in an input file with nonstandard labels. This field is initially set to zero. NSTD-REELS is tested each time an OPEN statement is executed for a file with nonstandard labels. If it is nonzero, its contents are used to determine the number of reels to be processed. If it is zero, the number of reels in the file is determined from the ASSIGN clause.

ORGANIZATION OF THE COBOL PROGRAM

Every COBOL source program is divided into four divisions. Each division must be placed in its proper sequence, and each must begin with a division header.

The four divisions, listed in sequence, and their functions are:

- IDENTIFICATION DIVISION, which names the program.
- ENVIRONMENT DIVISION, which indicates the machine equipment and equipment features to be used in the program.
- DATA DIVISION, which defines the nature and characteristics of data to be processed.
- PROCEDURE DIVISION, which consists of statements directing the processing of data in a specified manner at execution time.

Note: In all formats within this publication, the required clauses and optional clauses (when written) must appear in the sequence given in the format, unless the associated rules explicitly state otherwise.

Structure of the COBOL Program

{ IDENTIFICATION DIVISION. }

PROGRAM-ID. program-name.

[AUTHOR. [comment-entry]...]

[INSTALLATION. [comment-entry]...]

[DATE-WRITTEN. [comment-entry]...]

[DATE-COMPILED. [comment-entry]...]

[SECURITY. [comment-entry]...]

[REMARKS. [comment-entry]...]

ENVIRONMENT DIVISION.

{ CONFIGURATION SECTION.

SOURCE-COMPUTER. entry

OBJECT-COMPUTER. entry

[SPECIAL-NAMES. entry]

[INPUT-OUTPUT SECTION.

FILE-CONTROL. {entry}...

[I-O-CONTROL. entry]

DATA DIVISION.

[FILE SECTION.

{file description entry

{record description entry}...}...]

[WORKING-STORAGE SECTION.

{data item description entry}...

{record description entry}...]

[LINKAGE SECTION.

{data item description entry}...

{record description entry}...]

[REPORT SECTION.

{report description entry

{report group description entry}...}...]

PROCEDURE DIVISION USE {identifier-1} {identifier-2}...].

[DECLARATIVES.

{section-name SECTION. USE Sentence.

{paragraph-name. {sentence}...}...}...

END DECLARATIVES.]

{section-name SECTION {priority}.]

{paragraph-name. {sentence}...}...}...

Qualification

METHODS OF DATA REFERENCE

Every name used in a COBOL source program must be unique, either because no other name has the identical spelling, or because it is made unique through qualification, subscripting, or indexing.

An identifier is a data-name, unique in itself, or made unique by the syntactically correct combination of qualifiers, subscripts, and/or indexes.

QUALIFICATION

A name may be made unique if the name exists within a hierarchy of names and the name can be singled out by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers. Qualification is the process by which such a name is made unique.

Qualification is applied by placing after a data-name or a paragraph-name one or more phrases, each composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent. Only one qualifier is allowed for a paragraph-name.

Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. For example, if there is more than one file whose records contain the field EMPLOYEE-NO, yet there is but one file whose records are named MASTER-RECORD, EMPLOYEE-NO OF MASTER-RECORD would sufficiently qualify EMPLOYEE-NO. EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE is valid but unnecessary (see discussion of level indicators and level numbers in "Data Division").

The name associated with a level indicator is the highest level qualifier available for a data-name. (A level indicator (FD, SD, RD) specifies the beginning of a file description, sort file description, or report description.) A section-name is the highest (and the only) qualifier available for a procedure-name (see discussion of procedure-names in "Procedure Division"). Thus, level indicator names and section-names must be unique in themselves since they cannot be qualified.

The name of a conditional variable can be used as a qualifier for any of its condition-names. In addition, a conditional variable may be qualified to make it unique.

The rules for qualification follow:

1. Each qualifier must be of a successively higher level, and must be within the same hierarchy as the name it qualifies.
2. The same name must not appear at two levels in a hierarchy.
3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time reference is made to it in the Procedure, Environment, or Data Division (except in the REDEFINES clause where, by definition, qualification is unnecessary). (See the REDEFINES clause in "Data Division.")

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears.
5. A data-name cannot be subscripted when it is being used as a qualifier.
6. A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any of these combinations can be used.

Although user-defined data-names can be duplicated within the Data Division and Procedure Division, the following rules should be noted:

1. No duplicate section-names are allowed.
2. No data-name can be the same as a section-name or a paragraph-name.
3. Duplication of data-names must not occur in those places where the data-names cannot be made unique by qualification.

SUBSCRIPTING

Subscripts can be used only when reference is made to an individual element within a list or table of elements that have not been assigned individual data-names (see "Table Handling").

INDEXING

References can be made to individual elements within a table of elements by specifying indexing for that reference. An index is assigned to a given level of a table by using an INDEXED BY clause in the definition of the table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index (see "Table Handling").

Reference Format

USE OF THE COBOL CODING FORM

The reference format provides a standard method for writing COBOL source programs. The format is described in terms of character positions in a line on an input/output medium. Punched cards are the initial input medium to the COBOL compiler. The compiler only accepts source programs written in 80-column reference format (see Figure 5) and produces an output listing of the source program in the same reference format. 81-column input is not accepted.

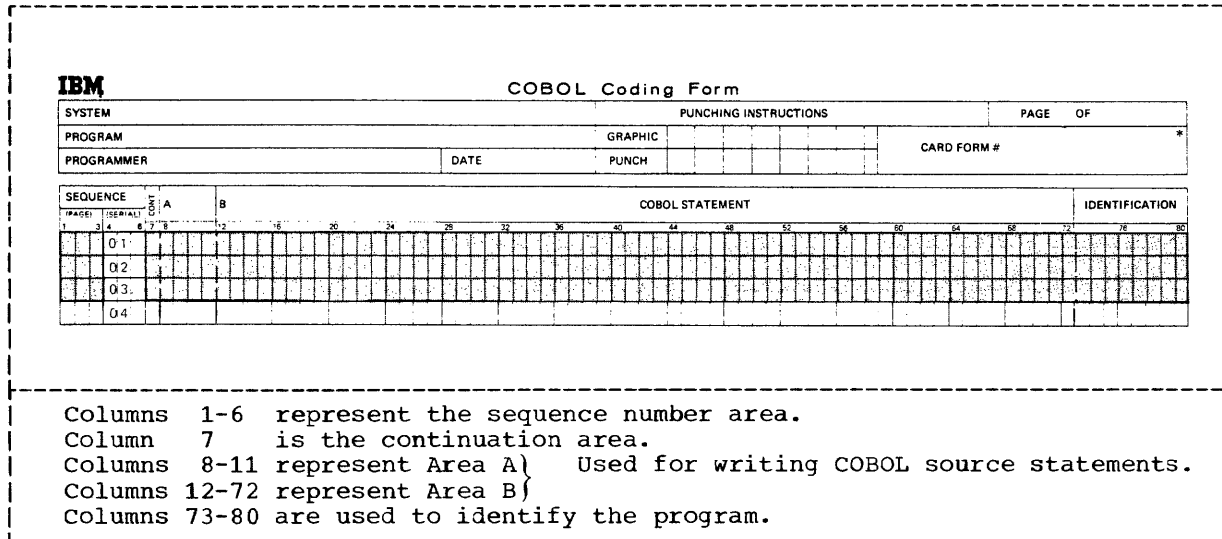


Figure 5. Reference Format

The rules for spacing given in the following discussion of the reference format take precedence over any other specifications for spacing given in this publication.

SEQUENCE NUMBERS

A sequence number, consisting of six digits in the sequence number area, is used to identify numerically each card image to be compiled by the COBOL compiler. The use of sequence numbers is optional.

If sequence numbers are present, they must be in ascending order. An error message is issued when source language input is out of sequence. **Sequence checking can be suppressed at compile time by overriding the compiler's default option of checking.**

AREA A AND AREA B

Area A, columns 8 through 11, is reserved for the beginning of division headers, section-names, paragraph-names, level indicators, and certain level numbers. Area B occupies columns 12 through 72.

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears.
5. A data-name cannot be subscripted when it is being used as a qualifier.
6. A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any of these combinations can be used.

Although user-defined data-names can be duplicated within the Data Division and Procedure Division, the following rules should be noted:

1. No duplicate section-names are allowed.
2. No data-name can be the same as a section-name or a paragraph-name.
3. Duplication of data-names must not occur in those places where the data-names cannot be made unique by qualification.

SUBSCRIPTING

Subscripts can be used only when reference is made to an individual element within a list or table of elements that have not been assigned individual data-names (see "Table Handling").

INDEXING

References can be made to individual elements within a table of elements by specifying indexing for that reference. An index is assigned to a given level of a table by using an INDEXED BY clause in the definition of the table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index (see "Table Handling").

Division Header

The division header must be the first line in a division. The division header starts in Area A with the division-name, followed by a space and the word DIVISION, and a period. ~~If this program is to be called, a space and a priority number may follow the words PROCEDURE DIVISION.~~ No other text may appear on the same line as the division header.

Section Header

The name of a section starts in Area A of any line following the division header. The section-name is followed by a space, the word SECTION, and a period. If program segmentation is desired, a space and a priority number may follow the word SECTION. No other text may appear on the same line as the section-header, except USE and COPY sentences.

Note: Although USE and COPY may appear in the Declaratives portion of the Procedure Division, only USE is restricted to the Declaratives portion. COPY may be used elsewhere in the COBOL program.

Paragraph-names and Paragraphs

The name of a paragraph starts in Area A of any line following the division header. It is followed by a period followed by a space.

A paragraph consists of one or more successive sentences. The first sentence in a paragraph begins anywhere in Area B of either the same line as paragraph-name or the immediately following line. Each successive line in the paragraph starts anywhere in Area B.

Level Indicators and Level Numbers

In those Data Division entries that begin with a level indicator, the level indicator begins in Area A followed in Area B by its associated file-name and appropriate descriptive information.

In those data description entries that begin with a level number 01 or 77, the level number begins in Area A followed in Area B by its associated data-name and appropriate descriptive information.

In those data description entries that begin with level numbers 02 through 49, 66, or 88, the level number may begin anywhere in Area A or Area B, followed in Area B by its associated data-name and descriptive information.

CONTINUATION OF LINES

Any sentence or entry that requires more than one line is continued by starting subsequent line(s) in Area B. These subsequent lines are called continuation lines. The line being continued is called the continued line. If a sentence or entry occupies more than two lines, all lines other than the first and last are both continuation and continued lines.

CONTINUATION OF NONNUMERIC LITERALS

When a nonnumeric literal is continued from one line to another, a hyphen is placed in column 7 of the continuation line, and a quotation mark preceding the continuation of the literal may be placed anywhere in Area B. All spaces at the end of the continued line and any spaces following the quotation mark of the continuation line and preceding the final quotation mark are considered part of the literal.

CONTINUATION OF WORDS AND NUMERIC LITERALS

When a word or numeric literal is continued from one line to another, a hyphen must be placed in column 7 of the continuation line to indicate that the first nonblank character in Area B of the continuation line is to follow the last nonblank character on the continued line, without an intervening space. In the case of numeric literals the last nonblank character of the continued line must not be a period or comma.

BLANK LINES

A blank line is one that contains nothing but spaces from column 7 through column 72, inclusive. A blank line may appear anywhere in the source program, except immediately preceding a continuation line.

COMMENT LINES

Explanatory comments may be inserted on any line within a source program by placing an asterisk in the Continuation Area of the line. Any combination of the characters from the EBCDIC set may be included in Areas A and B of that line. The asterisk and the characters will be produced on the source listing but serve no other purpose. (Also, see the NOTE statement in "Compiler Directing Statements" in "Procedure Division").

FORMAT NOTATION

Throughout this publication, basic formats are prescribed for various elements of COBOL. These generalized descriptions are intended to guide the programmer in writing his own statements. They are presented in a uniform system of notation, explained in the following paragraphs. Although it is not part of COBOL, this notation is useful in describing COBOL.

1. All words printed entirely in capital letters are reserved words. These are words that have preassigned meanings in COBOL. In all formats, words in capital letters represent an actual occurrence of those words. If any such word is incorrectly spelled, it will not be recognized as a reserved word and may cause an error in the program.
2. All underlined reserved words are required unless the portion of the format containing them is itself optional. These are key words. If any such word is missing or is incorrectly spelled, it is considered an error in the program. Reserved words not underlined may be included or omitted at the option of the programmer. These words are used only for the sake of readability; they are called optional words and, when used, must be correctly spelled.
3. The characters +, -, <, >, =, when appearing in formats, although not underlined, are required when such formats are used.
4. All punctuation and other special characters (except those symbols cited in the following paragraphs) represent the actual occurrence of those characters. Punctuation is essential where it is shown. Additional punctuation can be inserted, according to the rules for punctuation specified in this publication.
5. Words that are printed in lower-case letters represent information to be supplied by the programmer. All such words are defined in the accompanying text.
6. In order to facilitate references to them in text, some lower-case words are followed by a hyphen and a digit or letter. This modification does not change the syntactical definition of the word.
7. Certain entries in the formats consist of a capitalized word(s) followed by the word "Clause" or "Statement." These designate clauses or statements that are described in other formats, in appropriate sections of the text.
8. Square brackets ([]) are used to indicate that the enclosed item may be used or omitted, depending on the requirements of the particular program. When two or more items are stacked within brackets, one or none of them may occur.
9. Braces ({ }) enclosing vertically stacked items indicate that one of the enclosed items is obligatory.

Format Notation

10. The ellipsis (...) indicates that the immediately preceding unit may occur once, or any number of times in succession. A unit means either a single lower-case word, or a group of lower-case words and one or more reserved words enclosed in brackets or braces. If a term is enclosed in brackets or braces, the entire unit of which it is a part must be repeated when repetition is specified.

11. Comments, restrictions, and clarifications on the use and meaning of every format are contained in the appropriate portions of the text.

PART II -- IDENTIFICATION AND ENVIRONMENT DIVISIONS

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION -- FILE PROCESSING SUMMARY
- ORGANIZATION OF THE ENVIRONMENT DIVISION
- ENVIRONMENT DIVISION -- CONFIGURATION SECTION
- ENVIRONMENT DIVISION -- INPUT-OUTPUT SECTION

IDENTIFICATION DIVISION

The Identification Division is the first division of a COBOL program. It identifies the source program and the object program. A source program is the initial problem program; an object program is the output from a compilation.

In addition, the user may include the date the program is written, the date the compilation of the source program is accomplished, etc., in the paragraphs shown.

Structure of the Identification Division

```
{IDENTIFICATION DIVISION.
ID DIVISION}
```

```
PROGRAM-ID. program-name.
```

```
[AUTHOR. [comment-entry]...]
```

```
[INSTALLATION. [comment-entry]...]
```

```
[DATE-WRITTEN. [comment-entry]...]
```

```
[DATE-COMPILED. [comment-entry]...]
```

```
[SECURITY. [comment-entry]...]
```

```
[REMARKS. [comment-entry]...]
```

Specific paragraph-names identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional. If included, they must be presented in the order shown. ~~However, this compiler will accept them in any order.~~

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period. Each comment-entry may be any combination of characters from the EBCDIC set, organized to conform to sentence and paragraph structure. ~~This compiler will accept ID~~
~~Division followed by a period as a substitute for the standard division~~
~~header.~~

PROGRAM-ID Paragraph

The PROGRAM-ID paragraph gives the name by which a program is identified.

Format
<u>PROGRAM-ID.</u> program-name.

DATE-COMPILED Paragraph

The PROGRAM-ID paragraph contains the name of the program and must be present in every program.

Program-name identifies the object program to the control program. Program-name must conform to the rules for formation of a procedure-name. ~~However, the first eight characters of program-name are used as the identifying name of the program and should therefore be unique as a program-name.~~ The first eight characters of program-name are used as the identifying name of the program and should therefore be unique as a program-name.

Since the system expects the first character of program-name to be alphabetic, the first character, if it is numeric, will be converted as follows:

0 to J

1-9 to A-I

Since the system does not include the hyphen as an allowable character, the hyphen is converted to zero if it appears as the second through eighth character of the name.

Note: For additional information concerning program-name when using the Sort feature, the Segmentation feature, or the CATALR option, see the Programmer's Guide.

DATE-COMPILED Paragraph

The DATE-COMPILED paragraph provides the compilation date on the source program listing.

Format
<u>DATE-COMPILED.</u> [comment-entry]

The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a comment-entry is present, even though it spans lines, it is replaced in its entirety with the current date.

ENVIRONMENT DIVISION -- FILE PROCESSING SUMMARY

In COBOL, all aspects of the total data processing problem that depend on the physical characteristics of a specific computer are given in one portion of the source program known as the Environment Division. Thus, a change in computers entails major changes in this division only. The primary functions of the Environment Division are to describe the computer system on which the object program is run and to establish the necessary links between the other divisions of the source program and the characteristics of the computer.

The exact contents of the Environment Division depend on the method used to process files in the COBOL program. Before the language elements used in the Environment Division can be discussed meaningfully, some background in the file processing techniques available to the COBOL user must be given.

Each combination of data organization and access method specified in the COBOL language is defined as a file-processing technique. The file-processing technique to be used for a particular file is determined by the data organization of that file and whether the access method is sequential or random. Figure 6 summarizes the file-processing techniques.

DATA ORGANIZATION

Three types of data organization are made available to Disk Operating System COBOL users: sequential, direct, and indexed. The means of creating or retrieving logical records in a file differ, depending on which type of data organization exists (organization being the structure of data on a physical file). Each type of data organization is incompatible with the others. Organization of an input file must be the same as the organization of the file when it was created.

Sequential Data Organization

When sequential data organization is used, the logical records in a file are positioned sequentially in the order in which they are created and are read sequentially in the order in which they were created (or in sequentially reversed order if the REVERSED option of the OPEN statement is written for tape files). Such a file organization is referred to in this publication as standard sequential organization.

This type of data organization must be used for tape or unit-record files and may be used for files assigned to mass storage devices. No key is associated with records on a sequentially organized file.

Direct Data Organization

When direct data organization is used, the positioning of the logical records in a file is controlled by the user through the specification of an ACTUAL KEY defined in the Environment Division. The ACTUAL KEY has two components. The first is a track identifier which identifies the relative or actual track at which a record is to be placed or at which the search for a record is to begin. The second component is a record

Access Methods

identifier, which serves as a unique logical identifier for a specific record on the track. Files with direct data organization must be assigned to mass storage devices.

Indexed Data Organization

When indexed data organization is used, the position of each logical record in a file is determined by indexes maintained by the system and created with the file. The indexes are based on keys provided by the user. Indexed files must be assigned to mass storage devices.

ACCESS METHODS

Two access methods are available to users of DOS COBOL: sequential access and random access.

Sequential access is the method of reading and writing records of a file in a serial manner; the order of reference is implicitly determined by the position of a record in the file.

Random access is the method of reading and writing records in a programmer-specified manner; the control of successive references to the file is expressed by specifically defined keys supplied by the user.

ACCESSING A SEQUENTIAL FILE

A standard sequential file may be accessed only sequentially, i.e., records are read or written in order. Records can be created and retrieved; for standard sequential files on mass storage devices, records can also be updated.

ACCESSING A DIRECT FILE

Direct files may be accessed both sequentially and randomly. Records can be retrieved sequentially; they can be created, retrieved, updated, or added randomly.

Sequential Access

When reading a direct file sequentially, records are retrieved in logical sequence; this logical sequence corresponds exactly to the physical sequence of the records.

If the ACTUAL KEY clause is specified, the key associated with the record will be placed in the field specified by the ACTUAL KEY clause, for each READ statement executed.

Random Access

When accessing a direct file randomly, the ACTUAL KEY clause is required.

The system uses the ACTUAL KEY to determine which track a particular record is on and to locate the record on that track. If the record is found, the data portion of the record is read, or, for a rewrite operation, replaced by a new record. If during a READ operation, the desired record cannot be found on the specified track, an invalid key condition is said to exist.

Unless APPLY EXTENDED-SEARCH is used, only the specified track is searched for the desired record. When APPLY EXTENDED-SEARCH is used, the system searches the entire cylinder for the record if the desired record cannot be found.

For a write operation, the system, after locating the track, searches for the last record on the track, and writes the new record (with control fields, including a key field equal to the identifier found within the ACTUAL KEY field) after the last record.

When a direct file is being created, OPEN initializes the capacity records (RO) on all the tracks of the file. Therefore, a WRITE statement issued for an output file is processed in the same manner as a WRITE statement that adds a record to an input-output file.

ACCESSING AN INDEXED FILE

An indexed file may be accessed both sequentially and randomly. Records can be created, retrieved, or updated sequentially; they can be retrieved, updated, or added randomly.

The same indexed file may not be created and read or updated within the same program.

Sequential Access

An indexed file may only be created sequentially. When creating an indexed file, the RECORD KEY clause must be specified. It is used to indicate the location of the key within the record itself. The NOMINAL KEY clause may be specified. Records are arranged in the file in the order in which they are written.

To retrieve or update an indexed file sequentially, the RECORD KEY clause must be specified. Records are read in the order in which they are arranged in the file. Logically, this corresponds to the sequence of keys, which must be in collating sequence at the time the file is created. If record retrieval is to begin at other than the first record, the NOMINAL KEY clause must be specified and a START statement must be executed before the first READ operation for that file.

Random Access

To retrieve or update an indexed file randomly, both the NOMINAL KEY and RECORD KEY clauses are required. A record is considered "found" when the value associated with the data-name specified in the NOMINAL KEY clause is equal to the value of the RECORD KEY for the record. When

Access Methods

adding a record to or updating a randomly accessed indexed file, the value associated with the PRIMARY must be identical to that of NONPRIMARY KEY fields.

Appendix B contains three sample COBOL programs that illustrate:

1. Creation of a direct file

(Figure 4 contains a sample COBOL program illustrating random retrieval and updating of a direct file.)

2. Creation of an indexed file
3. Random retrieval and updating of an indexed file

DOS Organization	Device Type	Access	Organization
DTFCD	Reader	[SEQUENTIAL]	standard sequential
DTFCD	Punch	[SEQUENTIAL]	standard sequential
DTFPR	Printer	[SEQUENTIAL]	standard sequential
DTFMT	Tape	[SEQUENTIAL]	standard sequential
DTFSD	Mass Storage	[SEQUENTIAL]	standard sequential
DTFDA	Mass Storage	[SEQUENTIAL]	direct
DTFDA	Mass Storage	RANDOM	direct
DTFIS	Mass Storage	[SEQUENTIAL]	indexed
DTFIS	Mass Storage	RANDOM	indexed

Figure 6. Summary of File-Processing Techniques

ORGANIZATION OF THE ENVIRONMENT DIVISION

The Environment Division must begin in Area A with the heading ENVIRONMENT DIVISION followed by a period.

The Environment Division is divided into two sections: the Configuration Section and the Input-Output Section. When written, the sections and paragraphs must be in the sequence shown.

Structure of the Environment Division

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER paragraph

OBJECT-COMPUTER paragraph

[SPECIAL-NAMES paragraph]

[INPUT-OUTPUT SECTION.

FILE-CONTROL paragraph

[I-O-CONTROL paragraph]

SOURCE-COMPUTER Paragraph

ENVIRONMENT DIVISION -- CONFIGURATION SECTION

The Configuration Section deals with the overall specifications of computers. It is divided into three paragraphs: the SOURCE-COMPUTER paragraph, which describes the computer on which the source program is compiled; the OBJECT-COMPUTER paragraph, which describes the computer on which the object program (the program produced by the COBOL compiler) is executed; and, optionally, the SPECIAL-NAMES paragraph which relates the function-names used by the compiler to user-specified mnemonic-names.

General Format
<u>CONFIGURATION SECTION.</u> <u>SOURCE-COMPUTER.</u> source-computer-entry <u>OBJECT-COMPUTER.</u> object-computer-entry <u>[SPECIAL-NAMES.</u> special-names-entry]

Section-names and paragraph-names must begin in Area A.

The Configuration Section is the only section in the Environment Division that is optional.

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph serves only as documentation, and describes the computer upon which the program is to be compiled.

Format
<u>SOURCE-COMPUTER.</u> computer-name.

Computer-name may be specified as IBM-360[-model-number] or as IBM-370[-model-number].

The SOURCE-COMPUTER paragraph is treated as comments by the COBOL compiler.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph describes the computer on which the program is to be executed.

Format	
<u>OBJECT-COMPUTER.</u>	computer-name
[<u>MEMORY SIZE</u> integer	{ <u>WORDS</u> <u>CHARACTERS</u> <u>MODULES</u> }]
[<u>SEGMENT-LIMIT</u> IS priority-number].	

Computer-name is a word of the form IBM-360[-model-number].
Computer-name must be the first entry in the OBJECT-COMPUTER paragraph.

If the configuration implied by computer-name comprises more or less equipment than is actually needed by the object program, the MEMORY SIZE clause permits the specification of the actual subset (or superset) of the configuration.

The MEMORY SIZE clause is treated as comments by the COBOL compiler.

The SEGMENT-LIMIT clause is discussed in "Segmentation."

Program Product Information -- Version 3

Computer-name may also be specified as IBM-370[-model-number]. If IBM-370 is specified, System/370 instructions are generated by the compiler. When IBM-370 is specified, the object program must be executed on a System/370 machine.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph provides a means of relating function-names to user-specified mnemonic-names. The SPECIAL-NAMES paragraph can also be used to exchange the functions of the comma and the period in the PICTURE character string and in numeric literals. In addition, the user may specify a substitution character for the currency symbol (\$) in the PICTURE character string.

General Format	
<u>SPECIAL-NAMES.</u>	
[function-name-1 <u>IS</u> mnemonic-name]...	
[function-name-2 [<u>IS</u> mnemonic-name]	
{ <u>ON</u> STATUS <u>IS</u> condition-name-1	
{ <u>OFF</u> STATUS <u>IS</u> condition-name-2	
[<u>OFF</u> STATUS <u>IS</u> condition-name-2] }]...	
[<u>ON</u> STATUS <u>IS</u> condition-name-1] }	
[<u>CURRENCY</u> SIGN <u>IS</u> literal] [<u>DECIMAL-POINT</u> <u>IS</u> <u>COMMA</u>].	

When the SPECIAL-NAMES paragraph is specified, the comma or the semicolon may optionally be used to separate successive entries; there must be one, and only one, period, placed at the end of the paragraph.

Function-name-1 may be chosen from the following list:

- SYSLST
- SYSPCH
- SYSPUNCH
- SYSIPT
- CONSOLE
- C01 through C12
- CSP
- S01 through S05
- literal

If SYSLST, SYSPCH, SYSPUNCH, SYSIPT, or CONSOLE are specified, the associated mnemonic-names may be used in ACCEPT and DISPLAY statements. Each of these function-names may appear only once in the SPECIAL-NAMES paragraph.

If C01 through C12, CSP, S01 through S05 are specified, the associated mnemonic-names may be used in a WRITE statement with the BEFORE/AFTER ADVANCING option. These function-names are the carriage control characters shown in Figure 7.

Function-name-1	Action Taken
CSP	suppress spacing
C01 through C09	skip to channel 1 through 9, respectively
C10 through C12	skip to channel 10, 11, 12, respectively
S01 through S05	IBM 1442: pocket select 1 or 2 IBM 2540: pocket select P1 or P2 IBM 2560: stacker select 1 through 5 IBM 3525: stacker select 1 or 2 IBM 5425: stacker select 1 through 4

Figure 7. Choices of Function-name-1 and Action Taken

SPECIAL-NAMES Paragraph

The use of a literal indicates that function-name-1 identifies Report Writer output. The mnemonic-name should appear in a CODE clause in a report description entry (RD) (see "Report Writer"). One such SPECIAL-NAMES entry may be given for each report defined in a program. The specified literal must be a one-character nonnumeric literal.

Function-name-2 is used to define a one-byte switch and may be specified as UPSI-0 through UPSI-7. These switches represent the User Program Status Indicator bits in the DOS communications region (see IBM System/360 Disk Operating System: System Control and System Service Programs, Form C24-5036). The status of the switch is specified by a condition-name and interrogated by testing it. One condition-name may be associated with the ON status; another may be associated with the OFF status (see "Switch-Status Condition"). One condition-name must be associated with function-name-2. A mnemonic-name, a second condition-name, or both may be associated with the function-name-2 as well. The condition-names represent the equivalent of level-88 items where UPSI-n or mnemonic-name may be considered the conditional variable.

Either UPSI-n or mnemonic-name may be used for qualification of the condition-name.

Note: This compiler does not support hardware switches. However, the switch status function is provided as an extension to support the system UPSI switches, which are program switches that serve the function of hardware switches.

To use UPSI switches #2, you could code your program as follows:

```
SPECIAL NAMES.

    UPSI-2 IS SWITCH-02
    ON STATUS IS TAPE-FILE
    OFF STATUS IS DISK-FILE.

PROCEDURE DIVISION.

TEST-SWITCH-02.
    IF TAPE-FILE
    THEN GO TO OPEN-TAPE.
    IF DISK-FILE
    THEN GO TO OPEN-DISK.

(1) // UPSI 00100000
(2) // UPSI 00000000
```

When executing your program with the first UPSI card you would go to OPEN-TAPE; when executing it with the second UPSI card you would go to OPEN-DISK.

The literal which appears in the CURRENCY SIGN clause is used in the PICTURE clause to represent the currency symbol. The literal must be nonnumeric and is limited to a single character which must not be any of the following characters:

1. digits 0 through 9
2. alphabetic characters A, B, C, D, P, R, S, V, X, Z, or the space
3. special characters * + - , . ; () " or '

If the CURRENCY SIGN clause is not present, only the \$ can be used as the currency symbol in the PICTURE clause.

Note: If the CURRENCY SIGN clause specifies E, then floating point representation cannot be specified.

FILE-CONTROL Paragraph

The clause DECIMAL-POINT IS COMMA means that the function of the comma and the period are exchanged in PICTURE character strings and in numeric literals.

ENVIRONMENT DIVISION -- INPUT-OUTPUT SECTION

The Input-Output Section deals with the definition of each file, the identification of its external storage media, the assignment of the file to one or more input/output devices and with information needed for the most efficient transmission of data between the media and the object program. The section is divided into two paragraphs: the FILE-CONTROL paragraph, which names and associates the files used in the program with the external media; and the I-O-CONTROL paragraph, which defines special input/output techniques.

```
-----  
                          General Format  
-----  
[INPUT-OUTPUT SECTION.  
FILE-CONTROL. {file-control-entry} ...  
I-O-CONTROL. input-output-control-entry]]
```

FILE-CONTROL PARAGRAPH

Information that is used or developed by the program may be stored externally. File description entries in the Data Division name the files into which information is placed and specify their physical characteristics. The FILE-CONTROL paragraph assigns the files (by the names given in the file description entries) to input/output devices.

```
-----  
                          General Format  
-----  
FILE-CONTROL.  
    {SELECT Clause  
    ASSIGN Clause  
    [RESERVE Clause]  
    [FILE-LIMIT Clause]  
    [ACCESS MODE Clause]  
    [PROCESSING MODE Clause]  
    [ACTUAL KEY Clause]  
    [NOMINAL KEY Clause]  
    [RECORD KEY Clause]  
    [TRACK-AREA Clause].}...
```

Each SELECT sentence must begin with a SELECT clause followed immediately by an ASSIGN clause; the order in which the optional clauses are written is not significant.

The use of a literal indicates that function-name-1 identifies Report Writer output. The mnemonic-name should appear in a CODE clause in a report description entry (RD) (see "Report Writer"). One such SPECIAL-NAMES entry may be given for each report defined in a program. The specified literal must be a one-character nonnumeric literal.

Function-name-2 is used to define a one-byte switch and may be specified as UPSI-0 through UPSI-7. These switches represent the User Program Status Indicator bits in the DOS communications region (see IBM System/360 Disk Operating System: System Control and System Service Programs, Form C24-5036). The status of the switch is specified by a condition-name and interrogated by testing it. One condition-name may be associated with the ON status; another may be associated with the OFF status (see "Switch-Status Condition"). One condition-name must be associated with function-name-2. A mnemonic-name, a second condition-name, or both may be associated with the function-name-2 as well. The condition-names represent the equivalent of level-88 items where UPSI-n or mnemonic-name may be considered the conditional variable.

Either UPSI-n or mnemonic-name may be used for qualification of the condition-name.

Note: This compiler does not support hardware switches. However, the switch status function is provided as an extension to support the system UPSI switches, which are program switches that serve the function of hardware switches.

The literal which appears in the CURRENCY SIGN clause is used in the PICTURE clause to represent the currency symbol. The literal must be nonnumeric and is limited to a single character which must not be any of the following characters:

1. digits 0 through 9
2. alphabetic characters A, B, C, D, P, R, S, V, X, Z, or the space
3. special characters * + - , . ; () " or

If the CURRENCY SIGN clause is not present, only the \$ can be used as the currency symbol in the PICTURE clause.

Note: If the CURRENCY SIGN clause specifies P, then floating point representation cannot be specified.

The clause DECIMAL-POINT IS COMMA means that the function of the comma and the period are exchanged in PICTURE character strings and in numeric literals.

FILE-CONTROL Paragraph

ENVIRONMENT DIVISION -- INPUT-OUTPUT SECTION

The Input-Output Section deals with the definition of each file, the identification of its external storage media, the assignment of the file to one or more input/output devices and with information needed for the most efficient transmission of data between the media and the object program. The section is divided into two paragraphs: the FILE-CONTROL paragraph, which names and associates the files used in the program with the external media; and the I-O-CONTROL paragraph, which defines special input/output techniques.

```

                                     General Format
-----
[INPUT-OUTPUT SECTION.
FILE-CONTROL. {file-control-entry} ...
[I-O-CONTROL. input-output-control-entry]]

```

FILE-CONTROL PARAGRAPH

Information that is used or developed by the program may be stored externally. File description entries in the Data Division name the files into which information is placed and specify their physical characteristics. The FILE-CONTROL paragraph assigns the files (by the names given in the file description entries) to input/output devices.

```

                                     General Format
-----
FILE-CONTROL.
    {SELECT Clause
    ASSIGN Clause
    [RESERVE Clause]
    [FILE-LIMIT Clause]
    [ACCESS MODE Clause]
    [PROCESSING MODE Clause]
    [ACTUAL KEY Clause]
    [NOMINAL KEY Clause]
    [RECORD KEY Clause]
    [FILE-KEY Clause].}...

```

Each SELECT sentence must begin with a SELECT clause followed immediately by an ASSIGN clause; the order in which the optional clauses are written is not significant.

SELECT Clause

The SELECT clause is used to name each file in a program.

Format
<u>SELECT</u> [<u>OPTIONAL</u>] file-name

Each file used in the program must be named once and only once as a file-name following the key word SELECT.

Each file named in a SELECT clause must have a File Description (FD) entry or Sort File Description (SD) entry in the Data Division.

The key word OPTIONAL may be specified only for input files accessed sequentially. It is required for input files that are not necessarily present each time the object program is executed. When a file is not present at object time, the first READ statement for that file causes the imperative-statement following the key words AT END to be executed. However, OPTIONAL need not be specified and will be treated as a comment, since this function is performed through the ASSGN control statement with the IGN parameter.

ASSIGN Clause

The ASSIGN clause is used to assign a file to an external medium.

Format
<u>ASSIGN TO</u> [integer] system-name-1 [system-name-2] ...
[FOR <u>MULTIPLE</u> { <u>REEL</u> } 1 { <u>UNIT</u> }]

Integer indicates the number of input/output units for a given medium assigned to file-name. Since the number of units is determined at program execution time (see IBM System/360 Disk Operating System: System Control and System Service Programs, Order No. GC24-5036), the standard definition given above is not the action taken by this compiler.

When specified for files with standard labels or for unlabeled output tape files, the integer option is treated as comments. When integer is specified as greater than one for unlabeled input tape files, then at the end of every reel a message is issued to the operator asking whether or not end-of-file has been reached. It is the user's responsibility to provide the operator with correct information as to the number of reels in the file.

ASSIGN Clause

For multivolume input files with nonstandard labels, the integer option is required. For such files, the compiler is unable to distinguish between end-of-volume and end-of-file and, therefore, cannot determine the number of reels in the file. Therefore, for input files with nonstandard labels, the integer option is used to determine the number of reels in the file. Integer is an unsigned integer from 1 through 15. This number can be overridden by storing a nonzero integer in special register NSRD-REELS before opening the file. If NSRD-REELS is set to zero at the time the file is opened, integer is used. If integer is not specified, the system assumes that the file is contained on one reel.

All files used in a program must be assigned to an external medium. System-name specifies a device class, a particular device, the organization of data upon this device, and the external name of the file. Any system-names beyond the first are treated as comments.

FOR MULTIPLE REEL/UNIT is applicable whenever the number of tape units or mass storage devices assigned might be less than the number of reels or units in the file. However, this clause need not be specified. The system will automatically handle volume switching for sequentially accessed files. All volumes must be mounted for randomly accessed files. Therefore, when this clause is specified, it is treated as comments.

System-name has the following structure:

SYSnnn-class-device-organization[-name]

where:

nnn is a three-digit number from 000 through 240 inclusive. This field represents the symbolic unit to which the file is assigned.

class is a two-digit field that represents the device class. The allowable combinations of characters are:

DA for mass storage
UT for utility
UR for unit record

Files assigned to DA devices may have standard sequential or direct organization. When organization is direct, access may be either sequential or random.

Files assigned to DA devices may also have indexed organization. If indexed access is used, access may be either sequential or random.

Files assigned to UT or UR devices must have standard sequential organization.

device is a four- or five-digit field that represents a device number. Device number is used to specify a particular device within a device class.

The allowable devices for any given device class are as follows:

Mass storage (DA) 2311, 2314, 2321

Utility (UT) 2400, 2311, 2314, 2321

Unit record (UR) 1442R, 1442P, 1403, 1404 (continuous forms only), 1443, 2501, 2520R, 2520P, 2540R, 2540P

(R indicates reader, P indicates punch)

Note: Sort input, output, and work files may be assigned to any utility device except a 2321 (see "Sort").

organization is a one-character field that specifies file organization. The letters that may be specified for each type of file are as follows:

```

S   for standard sequential files
A   for direct files -- actual track addressing
D   for direct files -- relative track addressing
U   for direct files using REWRITE statement* -- actual track addressing
W   for direct files using REWRITE STATEMENT* -- relative track
    addressing
I   for indexed files

```

* When the file is opened as INPUT or OUTPUT, however, U is the equivalent of A, W is the equivalent of D.

Figure 8 can be used to determine the correct choice of the organization field in system-names.

name is a three- through seven-character field specifying the external-name by which the file is known to the system. If specified, it is the name that appears in the file-name field of the VOL, DLBL, or TLBL job control statement (see the appropriate Programmer's Guide). If name is not specified, the symbolic unit (SYSnnn) is used as the external-name. The field must be specified if more than one file is assigned to the same symbolic unit.

Note: An INPUT file must have the same file characteristics as it had when created. That is, file-dependent descriptions for the INPUT file -- such as the device and organization fields of the system-name, OPEN and CLOSE mode, record descriptions, and certain APPLY clauses -- must be consistent with those specified when the file was created.

ASSIGN Clause (Version 3)

Device Type	ACCESS	File Organization	Track Addressing	Organization Field in System-name
tape, punch, reader, printer	[SEQUENTIAL]	standard sequential	--	S
mass storage device	[SEQUENTIAL]	standard sequential	--	S
mass storage device	[SEQUENTIAL]	direct	actual	A
			relative	D
mass storage device	RANDOM	direct	actual	A
			relative	D
mass storage device	RANDOM	direct (REWRITE)	actual	U
			relative	W
mass storage device	[SEQUENTIAL]	indexed	--	I
mass storage device	RANDOM	indexed	--	I

Figure 8. Values of Organization Field for File Organization

Program Product Information -- Version 3

Note: ASCII considerations for the ASSIGN clause are given in Appendix E.

For Version 3, the following additional system devices are allowable:

Mass storage (DA) 2319, 3330
 Utility (UT) 2319, 3330, 3410, 3420
 Unit Record (UR) 2560P, 2560R, 2560W, 3211, 3504, 3505, 3525P, 3525R, 3525W, 3525M, 3881

For the Version 3 DA and UT devices (2319, 3330, 3410, 3420), as well as for the UR 3211 and 3881 devices, these numbers can be specified in the device field of system-name. For these devices, the valid entries for the other fields in system-name are unchanged.

For the 3504 and 3505 card readers, system-name has the following format:

$$\text{SYSnnn-UR-} \left\{ \begin{array}{l} 3504 \\ 3505 \end{array} \right\} - \left\{ \begin{array}{l} \text{S[R]} \\ \text{O} \end{array} \right\} \text{ [-name]}$$

The SYSnnn and name fields have the same valid entries as other devices.

For the 2560 MFCM, system-name has the following format:

$$\text{SYSnnn-UR-2560} \quad \left\{ \begin{array}{c} \text{P} \\ \text{R} \\ \text{W} \end{array} \right\} - \left\{ \begin{array}{c} \text{S} \\ \text{T} \\ \text{V} \\ \text{X} \\ \text{Y} \\ \text{Z} \end{array} \right\} \left\{ \begin{array}{c} \text{[P]} \\ \text{S} \end{array} \right\} \quad \text{[-name]}$$

The name field has the same valid entries as for other devices.

The SYSnnn field, for card files that do not utilize combined function processing, has the same valid entries as other devices.

The SYSnnn field has special considerations when combined function card processing is used. For each associated logical file within the combined function structure there must be a separate SELECT sentence; each such associated logical file must be specified with the same SYSnnn field. (See Appendix G: Combined Function Card Processing for a more detailed discussion.)

For the device field, the following entries are valid:

2560R, 3525R	for a card read file
2560P, 3525P	for a card punch file
2560W, 3525W	for a 1 to 6 line card print file
3525M	for a multiline card print file

For the organization field, depending on the device field, the following entries are valid:

3525R (reader)	}	S[R]	for sequential card read files
		V[R]	for read/print associated files
		X[R]	for read/punch/print associated files
		Y[R]	for read/punch associated files

Note: the optional R field specifies RCE (Read Column Eliminate) card reading. (See "RCE and OMR Format Descriptor" for further discussion.)

2560R (reader)	}	S[P]	for sequential card read files, primary input hopper
		SS	for sequential card read files, secondary input hopper
		V[P]	for read/print associated files, primary input hopper
		VS	for read/print associated files, secondary input hopper
		X[P]	for read/punch/print associated files, primary input hopper
		XS	for read/punch/print associated files, secondary input hopper
		Y[P]	for read/punch associated files, primary input hopper
		YS	for read/punch associated files, secondary input hopper

3525P (punch)	}	S	for sequential card punch files
		T	for punch-and-interpret files (see Note)
		X	for read/punch/print associated files
		Y	for read/punch associated files
Z	for punch/print associated files		

2560P (punch)	}	S[P]	for sequential card punch files, primary input hopper
		SS	for sequential card punch files, secondary input hopper
		T[P]	for punch-interpret files, primary input hopper
		TS	for punch-interpret files, secondary input hopper
		X[P]	for read/punch/print associated files, primary input hopper
		XS	for read/punch/print associated files, secondary input hopper
		Y[P]	for read/punch associated files, primary input hopper
		YS	for read/punch associated files, secondary input hopper
		Z[P]	for punch/print associated files, primary input hopper
ZS	for punch/print associated files, secondary input hopper		

Note: The T field denotes a normal punched output file for which the graphically printable punched characters are also printed on print lines 1 and 3 of the card. Line 1 contains the first 64 characters, left justified; line 3 contains the last 16 characters, right justified.

2560W (1 to 6 line print)	}	S[P]	for sequential print files, primary input hopper
		SS	for sequential print files, secondary input hopper
		V[P]	for read/print associated files, primary input hopper
		VS	for read/print associated files, secondary input hopper
		X[P]	for read/punch/print associated files, primary input hopper
		XS	for read/punch/print associated files, secondary input hopper
		Z[P]	for punch/print associated files, primary input hopper
		ZS	for punch/print associated files, secondary input hopper

3525W (2-line print)	}	W	for sequential 2-line print files
		V	for read/print associated files
		X	for read/punch/print associated files
		Z	for punch/print associated files

3525M (multi- line print)	}	S	sequential multiline print files
		V	for read/print associated files
		X	for read-punch-print associated files
		Z	for punch/print associated files

Note: All input hopper specifications for one associated file must be identical.

| Figure 9 has been deleted.

RCE AND OMR FORMAT DESCRIPTOR

When the user specifies O (for Optical Mark Read) or R (for Read Column Eliminate) in the organization field of system-name, then at object time he must provide a format descriptor as the first card(s) in his data deck. If the format descriptor is missing for such files, a message is issued to the operator, and the job is terminated.

The format descriptor must be the first card(s) in the data deck. Column 1 of the first card must be blank. The keyword FORMAT must be punched in columns 2 through 7. Column 8 must be blank. Columns 9 through 71 can contain the parameters that specify which columns of the data cards are to be read in OMR or RCE mode. Continuation cards are valid. A continuation code must be placed in column 72 of the preceding card. Parameters may then be continued, beginning in column 16 of the continuation card. Comments, if used, must follow the last operand on each card by at least one blank space, and continuation card restrictions must be observed.

The format of the format descriptor is as follows:

Col.

```

12....7.9.....
||      ||
||      ||
||      ||
VV      V V
  FORMAT (N1,N2)[,(N3,N4)]...

```

N1, N2, N3, and N4 may be any decimal integers from 1 through 80. However, N2 must be greater than or equal to N1. N4 must be greater than or equal to N3. In addition, for OMR processing, N1 and N2 must be both even or both odd, N3 and N4 must be both even or both odd, and N3 - N2 must be greater than or equal to 2.

In OMR mode, the user establishes which columns are to be read in OMR mode. For example, if the user wishes to read columns 1, 3, 5, 7, 9 and 70, 72, 74, 76, 78, 80 in OMR mode, the following format descriptor is valid:

FORMAT (1,9),(70,80)

In RCE mode, the user specifies those columns which are not to be read. For example, if the user chooses to eliminate columns 20 through 30, and columns 52 through 73, the following format descriptor is valid:

FORMAT (20,30),(52,73)

RESERVE Clause

The RESERVE clause allows the user to modify the number of input/output areas (buffers) allocated by the compiler for a standard sequential file or (Version 3 only) a sequentially accessed indexed file.

Format			
<u>RESERVE</u>	{ integer }	ALTERNATE	[AREA]
	{ NO }		[AREAS]

This clause may be specified only for a standard sequential file.

Integer must have a value of 1.

A minimum of one buffer is required for a file. If this clause is omitted or if 1 is specified, one additional buffer is assumed.

If NO is specified, no additional buffer areas are reserved aside from the minimum of one.

Program Product Information -- Version 3

For Version 3, the RESERVE clause may specify one additional buffer for a sequentially accessed indexed file.

Combined function file processing considerations for the RESERVE clause are given in Appendix G.

FILE-LIMIT Clause

The FILE-LIMIT clause serves only as documentation, and is used to specify the beginning and the end of a logical file on a mass storage device.

Format					
{ <u>FILE-LIMIT IS</u> }	{ data-name-1 }		<u>THRU</u>	{ data-name-2 }	
{ <u>FILE-LIMITS ARE</u> }	{ literal-1 }			{ literal-2 }	
	{ data-name-3 }		<u>THRU</u>	{ data-name-4 }	
[{ literal-3 }			{ literal-4 }] ...

ACCESS MODE/PROCESSING MODE Clauses

The logical beginning of a mass storage file is the address specified as the first operand of the FILE-LIMIT clause; the logical end of a mass storage file is the address specified as the last operand of the FILE-LIMIT clause. Because file boundaries are determined at execution time from the control cards, this clause need not be specified and will be treated as comments.

ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which records of a file are to be accessed.

Format	
<u>ACCESS MODE</u> <u>IS</u>	{ <u>SEQUENTIAL</u> <u>RANDOM</u> }

If this clause is not specified, ACCESS IS SEQUENTIAL is assumed. For ACCESS IS SEQUENTIAL, records are placed or obtained sequentially. That is, the next logical record is made available from the file when the READ statement is executed, or the next logical record is placed into the file when a WRITE statement is executed. ACCESS IS SEQUENTIAL may be applied to files assigned to tape, unit-record, or mass storage devices.

For ACCESS IS RANDOM, storage and retrieval are based on an ACTUAL KEY ~~for each record~~ associated with each record. When the RANDOM option is specified, the file must be assigned to a mass storage device. ACCESS IS RANDOM may be specified when file organization is direct ~~or indexed~~.

The keyword IS must be specified. ~~_____~~

PROCESSING MODE Clause

The PROCESSING MODE clause serves only as documentation, and indicates the order in which records are processed.

Format	
<u>PROCESSING MODE</u> <u>IS</u> <u>SEQUENTIAL</u>	

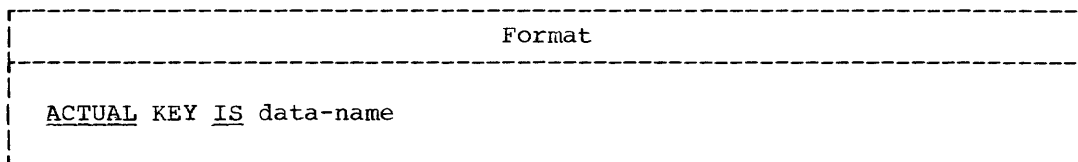
This clause is treated as comments, and may be omitted.

ACTUAL KEY Clause

An ACTUAL KEY is a key that is directly usable by the system to locate a logical record on a mass storage device. The ACTUAL KEY is made up of two components.

1. The track identifier, which expresses a track address at which the search for a record, or for a space in which to place a new record, is to begin.
2. The record identifier, which serves as a unique identifier for the record and is associated with the record itself.

When processing a randomly accessed direct file, the programmer is responsible for providing the ACTUAL KEY for each record to be processed.



Records are accessed randomly and are processed in the order in which they are accessed.

The ACTUAL KEY clause must be specified for direct files when ACCESS IS RANDOM is specified.

The ACTUAL KEY clause may be specified when reading direct files sequentially.

When a SEEK statement is executed, the contents of data-name are used to locate a specific mass storage record area.

When a READ statement is executed, a specific logical record (located by the system using the contents of data-name) is made available from the file.

When a WRITE statement is executed, the given logical record is written at a specific location in the file.

At file creation time, when no more room remains on a given track, a standard error occurs, and the user must provide a USE AFTER STANDARD ERROR declarative routine to update the track address.

The keyword IS must be specified.

The location of a particular logical record must be placed in data-name before the execution of the SEEK statement (or if no SEEK statement is present, the READ and WRITE statements).

Data-name must be a fixed-length item. It must be defined in the File Section, the Working-Storage Section, or the Data Base Section. However, if data-name is specified in the File Section it may not be contained in the file for which it is the key. Data-name is made up of two components: the track identifier, and the record identifier.

ACTUAL KEY Clause

TRACK IDENTIFIER: The track identifier may be expressed in two ways -- through relative track addressing, or through actual track addressing.

Relative Track Addressing: The track identifier is used to specify the relative track address at which a record is to be placed, or at which the search for a record is to begin.

Track identifier must be 4 bytes in length, and must be defined as a 32-bit integer binary data item whose maximum value does not exceed 16,777,215.

Actual Track Addressing: The track identifier is used to specify the actual track address at which a record is to be placed, or at which the search for a record is to begin.

Track identifier must be a binary data item eight bytes in length. No conversion is made by the compiler when determining the actual track address. The structure of these eight bytes and the permissible specifications are shown in Figure 10.

Before beginning processing, it is the user's responsibility to initialize R to the figurative-constant LOW-VALUE. The user need not concern himself further with this field.

		PACK	CELL		CYLINDER		HEAD		RECORD
		M	B	B	C	C	H	H	R
Device	Byte	0	1	2	3	4	5	6	7
2311		0-221	0	0	0	0-199	0	0-9	0-255
2314		0-221	0	0	0	0-199	0	0-19	0-255
2321		0-221	0	0-9	0-19	0-9	0-4	0-19	0-255
2319 (Version 3)		0-221	0	0	0	0-199	0	0-19	0-255
3330 (Version 3)		0-221	0	0	0-403		0-18		0-255

Figure 10. Structure of the First Eight Bytes of ACTUAL KEY -- Actual Track Addressing

RECORD IDENTIFIER: The symbolic portion of ACTUAL KEY used to identify a particular record on a track is the record identifier.

Record identifier must be from 1 through 255 bytes in length. Data within these bytes is treated exactly as specified.

A record is considered "found" when, for a given track, the record identifier at retrieval time matches the record identifier of a record in the file being searched.

ACTUAL KEY EXAMPLES: Two examples follow, to represent the coding necessary to specify the data-name in the ACTUAL KEY clause.

Relative Track Addressing: The following example shows an ACTUAL KEY using relative track addressing:

```

ENVIRONMENT DIVISION.
.
.
.
  ACTUAL KEY IS THE-ACTUAL-KEY.
.
.
.
DATA DIVISION.
.
.
.
WORKING-STORAGE SECTION.
01 THE ACTUAL-KEY.
   02 RELATIVE-TRACK-KEY USAGE COMPUTATIONAL PICTURE IS S9(8)
      VALUE IS 10 SYNCHRONIZED.
   02 EMPLOYEE-NO PICTURE IS X(6) VALUE IS LOW-VALUE.

```

Actual Track Addressing: The following example shows an ACTUAL KEY using actual track addressing:

```

ENVIRONMENT DIVISION.
.
.
.
  ACTUAL KEY IS THE-ACTUAL-KEY.
.
.
.
DATA DIVISION.
.
.
.
WORKING-STORAGE SECTION.
01 BINARY-FIELD-1.
   05 TRACK-ID.
      10 M USAGE COMPUTATIONAL PICTURE S999 VALUE IS 0.
      10 BB USAGE COMPUTATIONAL PICTURE S9 VALUE IS 0.
      10 CC USAGE COMPUTATIONAL PICTURE S999 VALUE IS 10.
      10 HH USAGE COMPUTATIONAL PICTURE S999 VALUE IS 0.
      10 R PICTURE IS X VALUE IS LOW-VALUE.
   05 EMPLOYEE-NO PICTURE XXXXXX VALUE IS LOW-VALUES.
01 ACTUAL-FIELD-1 REDEFINES BINARY-FIELD-1.
   05 FILLER PICTURE IS X.
   05 THE-ACTUAL-KEY PICTURE IS X(14).

```

Although the track identifier field must consist of eight bytes, nine bytes are defined within TRACK-ID. This is because the entry

```
10 M USAGE COMPUTATIONAL PICTURE S999
```

necessarily defines two bytes. However, as Figure 4 shows, the M field must be one byte in length. Therefore, BINARY-FIELD-1 must be redefined as ACTUAL-FIELD-1. In this way the superfluous high-order M byte can be stripped off from THE-ACTUAL-KEY through specification of the entry

```
05 FILLER PICTURE IS X
```

in ACTUAL-FIELD-1. The first eight bytes of THE-ACTUAL-KEY thus represent the track identifier, and the last six bytes represent the record identifier.

NOMINAL KEY Clause

NOMINAL KEY Clause

A NOMINAL KEY is used with indexed files. The clause specifies a symbolic identity for a specific logical record.

Format

NOMINAL KEY IS data-name

A NOMINAL KEY clause is required when an indexed file is accessed randomly, or when an indexed file is accessed sequentially and a START statement is used.

When the NOMINAL KEY clause is specified for an indexed file that is accessed randomly:

Data-name may specify any fixed-length Working-Storage item from 1 through 255 bytes in length.

Data-name must be at a fixed displacement from the beginning of the record description in which it appears; that is, it may not appear in the entry subsequent to an OCCURS clause with a DEPENDING ON option.

The symbolic identity of the record must be placed in data-name before the execution of the READ, WRITE, or REWRITE statement.

The symbolic identity is used when retrieving or updating a record to locate the logical record with a matching RECORD KEY, or, when adding a record, to create the key that will be associated with the record.

When a READ statement is executed, a specific logical record is made available from the file, using the contents of data-name.

When a WRITE or REWRITE statement is executed, the symbolic identity of the record specified by data-name is used to determine the physical location at which the record is written.

When the NOMINAL KEY is specified for an indexed file that is accessed sequentially:

Data-name may specify any fixed-length Working-Storage item from 1 through 255 bytes in length.

Data-name must be at a fixed displacement from the beginning of the record description in which it appears; that is, it may not appear in the entry subsequent to an OCCURS clause with a DEPENDING ON option.

The NOMINAL KEY clause must be specified if the START statement is used. When the START statement is executed, the contents of data-name are used to locate the record at which processing is to begin. The next READ statement accesses this record.

RECORD KEY Clause

A RECORD KEY is used to access an indexed file. It specifies the item within the data record that contains the key for the record.

Format

RECORD KEY IS data-name

The RECORD KEY clause must be specified for an indexed file.

Data-name must be a fixed-length item within the record. It must be less than 256 bytes in length.

When more than one record description is associated with a file, an identical field must appear in each description and must be in the same relative position from the beginning of the record, although the same data-name need not be used.

For compatibility with other systems, data-name should be defined to exclude the first byte of the record in the following cases:

1. A file with unblocked records
2. A file from which records are to be deleted
3. A file in which any key starts with a delete-code character (HIGH-VALUE)

With these exceptions, the item specified by data-name may appear anywhere within the record.

TRACK-AREA Clause

This clause may be optionally used when records are to be added to an indexed file in the random access mode. Efficiency in adding a record is improved when this clause is specified.

Format

TRACK-AREA IS integer CHARACTERS

The size of the area may be defined to hold from one to all the blocks on a track including their count and key fields.

Integer must be at least $24 + N$ ($40 + \text{RECORD KEY length} + \text{block size}$), where N is any number from 2 to the maximum number of blocks on a track. If N equals 1, then integer must be $24 + 50 + \text{RECORD KEY length} + \text{block size}$. Integer must not exceed 32,767. (See IBM System/360 Disk Operating System: Data Management Concepts, Form C24-3427).

RERUN Clause

I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph defines some of the special techniques to be used in the program. It specifies the points at which checkpoints are to be established, the core storage area which is to be shared by different files, the location of files on multiple-file reels, and optimization techniques. The I-O-CONTROL paragraph and its associated clauses are an optional part of the Environment Division.

```
-----  
                          General Format  
-----  
  
I-O-CONTROL.  
  [RERUN Clause] ...  
  [SAME AREA Clause] ...  
  [MULTIPLE FILE TAPE Clause] ...  
  [APPLY Clause] ... .  
-----
```

The order in which the I-O-CONTROL paragraph clauses are written is not significant.

RERUN Clause

The presence of a RERUN clause specifies that checkpoint records are to be taken. A checkpoint record is a recording of the status of a problem program and main storage resources at desired intervals. The contents of core storage are recorded on an external storage device at the time of the checkpoint and can be read back into core storage to restart the program from that point.

```
-----  
                          Format 1  
-----  
  
RERUN ON system-name  
  
  EVERY integer RECORDS OF file-name  
-----
```

```
-----  
                          Format 2  
-----  
  
RERUN ON system-name  
-----
```

The system-name in the RERUN clause specifies the external medium for the checkpoint file, the file upon which checkpoint records are to be written. It has the following structure:

```
SYSnnn-class-device-organization[-name]
```

The SYSnnn and name fields in the system-name for the checkpoint file cannot be the same as any specified in any ASSIGN clause.

Checkpoint records are written sequentially, and may be assigned to any utility or mass storage device (except the 2321). Only one RERUN clause in a program may use a mass storage device for writing checkpoint records. (A complete list of utility and mass storage devices is given in the description of system-name in the ASSIGN clause.)

Format 1 specifies that checkpoint records are to be written on the unit specified by system-name for every integer records of file-name that are processed. The value of integer must not exceed 16,777,215.

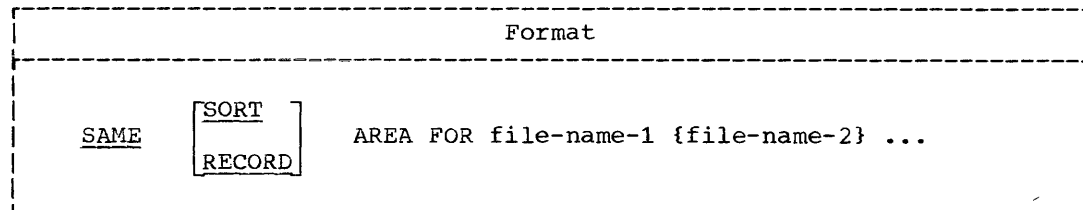
More than one Format 1 RERUN clause may be included in a program. If multiple RERUN clauses are specified, they may be specified either for the same or for different checkpoint files.

Format 2 is used for checkpointing sort-files and is described in "Sort."

Note: ASCII considerations for the RERUN clause are given in Appendix E.

SAME Clause

The SAME clause specifies that two or more files are to use the same core storage during processing.



A SAME clause with the SORT option is described in "Sort." The following discussion pertains only to the SAME AREA and SAME RECORD AREA clauses.

The SAME RECORD AREA clause specifies that two or more files are to use the same main storage for processing the current logical record. All of the files may be open at the same time. A logical record in the shared storage area is considered to be:

- a logical record of each opened output file in this SAME RECORD AREA clause, and
- a logical record of the most recently read input file in this SAME RECORD AREA clause.

If the SAME clause does not contain the RECORD option, the area being shared includes all storage areas assigned to the files; therefore, it is not valid to have more than one of these files open at one time.

MULTIPLE FILE TAPE Clause

More than one SAME clause may be included in a program; however:

1. A file-name must not appear in more than one SAME AREA clause.
2. A file-name must not appear in more than one SAME RECORD AREA clause.
3. If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, that SAME RECORD AREA clause may contain additional file-names other than those that appear in that SAME AREA clause.

The SAME RECORD AREA clause implicitly redefines the logical records of each file named. This allows the user to write the same record to more than one file, or to write a record he has just read without any extra MOVE statement. An extra record area is generated for this purpose.

The SAME AREA clause saves space generated for record areas. However, files named in a SAME AREA clause cannot be open at the same time, thus limiting processing possibilities.

MULTIPLE FILE TAPE Clause

The MULTIPLE FILE TAPE clause is used to indicate that two or more files share the same physical reel of tape.

```
Format
-----
MULTIPLE FILE TAPE CONTAINS file-name-1
    [POSITION integer-1] [file-name-2 [POSITION integer-2]] ...
```

The MULTIPLE FILE TAPE clause is required when more than one file shares the same physical reel of tape. However, this compiler permits the clause to be omitted when:

1. The files have standard labels
2. Each file is processed in the sequence in which it appears on the reel and a rewind is not executed for any file on the reel except the last file to be processed

The MULTIPLE FILE TAPE clause is pertinent only when the tape has nonstandard labels, or when labels are omitted. It is treated as comments for a tape that has standard labels.

Regardless of the number of files on a single reel, only those files that are used in the object program need be specified.

For purposes of positioning, a physical file is considered to be that segment of a tape that is terminated by a tape mark. Note that two consecutive tape marks are considered to terminate two physical files.

If all file-names refer to single physical files and have been listed in consecutive order, the POSITION option need not be given.

The POSITION integer relative to the beginning of the tape must be given if any file on the tape is not listed, or if a tape contains more than one physical file, i.e., more than one tape mark. Therefore, if a tape contains two files, each having one nonstandard header label

APPLY Clause

terminated by a tape mark, their positions would be 1 and 3. If the labels are not to be processed, the positions may be specified as 2 and 4, and the LABEL RECORDS clause must specify OMITTED.

The compiler will position the tape by skipping past a number of tape marks equal to POSITION number minus one.

Caution: POSITION should be used only for input files. If POSITION is used for output files, overlay may occur.

More than one MULTIPLE FILE clause may be included in a program.

APPLY Clause

There are six options of the APPLY clause. Any or all may appear in one COBOL program.

Format for Option 1

APPLY WRITE-ONLY ON file-name-1 [file-name-2] ...

This option is used to make optimal use of buffer space allocated when creating a standard sequential file with blocked V-mode records. Normally, a buffer is truncated when the maximum size record no longer fits. Use of this option will cause a buffer to be truncated only when the next record does not fit in the buffer. The file may be opened only for OUTPUT.

Every WRITE statement associated with the file must use the WRITE record-name FROM identifier option. None of the subfields of record-name may be referred to in procedural statements, nor may any of the subfields be the object of an OCCURS DEPENDING ON clause.

Format for Option 2

APPLY EXTENDED-SEARCH ON file-name-1 [file-name-2] ...

This option is used to control the extent of the search made for a specified record. It may refer only to direct files opened as INPUT or I-O when ACCESS IS RANDOM. In normal operation, execution of a READ or REWRITE statement for a file causes the specified track to be searched. If the desired record cannot be found on the specified track, the INVALID KEY option of the READ or REWRITE statement is executed.

If EXTENDED-SEARCH is applied, the search for a specified record key begins on the track specified and continues until one of two conditions occurs:

1. The record is found.
2. The end of the specified cylinder is reached.

In the second case, the INVALID-KEY option of the READ or REWRITE is executed. The INVALID KEY option may also be executed when the specified

APPLY Clause

track is outside the limits of the file. It is the user's responsibility to determine the cause of the INVALID KEY condition.

Format for Option 3

```
APPLY WRITE-VERIFY ON file-name-1 [file-name-2] ...
```

This option is used to cause parity checking of records written on a mass storage device. WRITE-VERIFY is always assumed when records are written on a 2321 mass storage device.

If this clause is omitted, any records written on the mass storage devices are not verified. This clause may be specified for any file assigned to any mass storage device except the 2321.

Format for Option 4

```
APPLY CYL-OVERFLOW OF integer TRACKS ON file-name-1  
[file-name-2] ...
```

This option may be specified when records are being created for or added to an indexed file. It specifies the number of tracks to be reserved for overflow records on each cylinder.

The maximum value of integer for each valid device, plus the number of tracks reserved when the APPLY clause is omitted is as follows:

<u>Device</u>	<u>Maximum Integer Value</u>	<u>Default</u>
2311	8	2
2314	18	4
2319 (Version 3)	18	4
3330 (Version 3)	17	3

If integer is specified as 0, no cylinder overflow area is reserved.

APPLY Clause

Format for Option 5

```

APPLY { MASTER-INDEX
          } TO device-number ON file-name-1
        { CYL-INDEX
          }
[file-name-2] ...

```

This option may be specified only for indexed files. It is used to specify the highest level index and the device on which it is to reside.

Device-number must specify a mass storage device. If this clause is omitted for an indexed file, cylinder index is assumed to be the highest level index and the index must reside on the same device as the prime data. If MASTER-INDEX is specified, then both the master index and the cylinder index will reside on the device specified.

Format for Option 6

```

APPLY CORE-INDEX TO data-name ON file-name-1 [file-name-2] ...

```

This option may be specified only for an indexed file being accessed randomly. It specifies the storage area to be used to hold the cylinder index in core. The cylinder index contains one entry for each cylinder in the prime data area. All or part of the cylinder index can reside in main storage. If the core storage area defined by data-name is large enough for all the cylinder index entries to be read into storage at one time, no presorting of the record keys need be done. If, however, the area assigned to the cylinder index is not large enough, the keys of the records to be processed should be presorted so that the cylinder index in core is fully utilized.

Data-name must be a fixed-length item defined in the Working-Storage Section. The area defined by data-name must be at least $(3+N)*(6+\text{key length})$, where N is the number of cylinder index entries to be read into core storage at one time, and key length is the length of the record key. In order to hold the entire index in core, N must equal the number of cylinders in the prime data area. Data-name must be at a fixed displacement from the beginning of the record description entry in which it appears, that is, it may not appear in the entry subsequent to an OCCURS clause with the DEPENDING ON option.

If multiple file-names are specified for the same data-name, only one of these files can be open at any given time. A file-name must not be specified in more than one APPLY CORE-INDEX clause. When using the REDEFINES clause, care should be taken that two data-names that refer to the same core storage area are not associated with two files that are open at the same time.

Data-name should not be referred to by Procedure Division statements when any file associated with it is open.

PART III -- DATA DIVISION

- DATA DIVISION -- INTRODUCTION

- ORGANIZATION OF THE DATA DIVISION

- FILE DESCRIPTION ENTRY -- DETAILS OF CLAUSES

- DATA DESCRIPTION

- DATA DESCRIPTION -- DETAILS OF CLAUSES



DATA DIVISION -- INTRODUCTION

The Data Division of a COBOL source program contains the description of all information to be processed by the object program. Two types of data may be processed by a COBOL program: information recorded externally on files and information created internally. The second type, which exists only during the execution of a program, will be discussed later in this chapter in "Working-Storage Section."

ORGANIZATION OF EXTERNAL DATA

A file is a collection of records. There are two types of records: physical records and logical records. A physical record is a group of characters or records which is treated as an entity when moved into or out of core storage. A logical record is a number of related data items. It may itself be a physical record, i.e., contained within a single physical unit, or it may be one of several logical records contained within a single physical unit, or it may extend across two or more physical units.

COBOL source language statements provide the means of describing the relationship between physical and logical records. Once this relationship is established, only logical records are made available to the COBOL programmer. Hence, in this publication, a reference to records means logical records unless the term "physical records" is used.

DESCRIPTION OF EXTERNAL DATA

In the discussion of data description, a distinction must first be made between a record's external description and its internal content.

External description refers to the physical aspects of a file, i.e., the way in which the file appears on an external medium. For example, the number of logical records per physical record describes the grouping of records in the file. The physical aspects of a file are specified in File Description entries.

A COBOL record usually consists of groups of related information that are treated as an entity. The explicit description of the contents of each record defines its internal characteristics. For example, the type of data to be contained within each field of a logical record is an internal characteristic. This type of information about each field of a particular record is grouped into a Record Description entry.

ORGANIZATION OF THE DATA DIVISION

The Data Division is divided into four sections: the File Section, the Working-Storage Section, the Linkage Section, and the Report Section.

All data that is stored externally, for example, on magnetic tape, must be described in the File Section before it can be processed by a COBOL program. Information that is developed for internal use must be described in the Working-Storage Section. Information passed from one program to another must be described in the Linkage Section. The content and format of all reports that are to be generated by the Report Writer feature must be described in the Report Section.

The Data Division is identified by, and must begin with, the header DATA DIVISION. The File Section is identified by, and must begin with, the header FILE SECTION. The header is followed by one or more file description entries and one or more associated record description entries. The Working-Storage Section is identified by, and must begin with, the header WORKING-STORAGE SECTION. The header is followed by data item description entries for noncontiguous items, followed by record description entries. The Linkage Section is identified by, and must begin with, the header LINKAGE SECTION. The header is followed by noncontiguous data item description entries, followed by record description entries. The Report Section is identified by, and must begin with, the header REPORT SECTION. The header is followed by one or more report description entries, and one or more report group description entries.

For the proper formats of Division and Section headers, see "Use of the COBOL Coding Form" in "Language Considerations."

Structure of the Data Division

DATA DIVISION.

FILE SECTION.

{file description entry
{record description entry}...}...

WORKING-STORAGE SECTION.

[data item description entry]...
[record description entry]...

LINKAGE SECTION.

[data item description entry]...
[record description entry]...

REPORT SECTION.

{report description entry
{report group description entry}...}...

Each of the sections of the Data Division is optional and may be omitted from the source program when the section is unnecessary. When used, the sections must appear in the foregoing sequence.

ORGANIZATION OF DATA DIVISION ENTRIES

Each Data Division entry begins with a level indicator or a level number, followed by one or more spaces, followed by the name of a data item (except in the Report Section), followed by a sequence of independent clauses describing the data item. The last clause is always terminated by a period followed by a space.

Level Indicator

The level indicator FD is used to specify the beginning of a file description entry. When the file is a sort-file, the level indicator SD must be used instead of FD (see "Sort"). When a report is to be generated by the Report Writer feature, the level indicator RD, specifying the beginning of a report description entry must be provided for each report in addition to the FD for the file on which the report is generated (see "Report Writer"). Figure 11 summarizes the level indicators.

Indicator	Use
FD	File description entries
SD	Sort-file description entries
RD	Report description entries

Figure 11. Level Indicator Summary

Each level indicator must begin in Area A and be followed in Area B by its associated file-name and appropriate descriptive information.

Level indicators are illustrated in the sample COBOL programs found in Appendix B.

Level Number

Level numbers are used to structure a logical record to satisfy the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data reference.

The basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record may consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. A group item consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into larger groups. Thus, an elementary item may belong to more than one

Level Number

group. In the following example, the group items MARRIED and SINGLE are themselves part of a larger group named RETIRED-EMPLOYEES:

```
02 RETIRED-EMPLOYEES.
  03 MARRIED.
    04 NO-MALE PICTURE 9(8).
    04 NO-FEMALE PICTURE 9(8).
  03 SINGLE.
    04 NO-MALE PICTURE 9(8).
    04 NO-FEMALE PICTURE 9(8).
```

A system of level numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, the level number for a record must be 1 or 01. Less inclusive data items are assigned higher (not necessarily successive) level numbers not greater than 49. There are special level numbers -- 66, 77, and 88 -- which are exceptions to this rule. Separate entries are written in the source program for each level number used.

A group includes all group and elementary items following it until a level number less than or equal to the level number of that group is encountered. The level number of an item which immediately follows the last elementary item of the previous group must be equal to the level number of one of the groups to which a prior elementary item belongs.

Standard

```
01 A.
  05 C-1.
    06 D PICTURE X.
    06 E PICTURE X.
  05 C-2.
  .
  .
  .
```

Nonstandard

```
01 A.
  05 C-1.
    06 D PICTURE X.
    06 E PICTURE X.
  05 B-1.
```

In the foregoing example, this compiler will accept the nonstandard

Level numbers 01 and 77 must begin in Area A, followed in Area B by associated data names and appropriate descriptive information. All other level numbers may begin in either Area A or in Area B, followed in Area B by associated data names and appropriate descriptive information.

A single-digit level number is written either as a space followed by a digit or as a zero followed by a digit. At least one space must separate a level number from the word following the level number.

Special Level Numbers

Three types of data exist whose level numbers are not intended to structure a record. They are:

- 66: Names of elementary items or groups described by a RENAMES clause for the purpose of regrouping data items have been assigned the special level number 66. For an example of the function of the RENAMES clause, see "Data Description."
- 77: Noncontiguous Working-Storage items, which are not subdivisions of other items and are not themselves subdivided, have been assigned the special level number 77.

88: Entries that specify condition-names to be associated with particular values of a conditional variable have been assigned the special level number 88. For an example of level-88 items, see "Data Description."

Indentation

Successive data description entries may have the same format as the first such entry or may be indented according to level number. Indentation is useful for documentation purposes, and does not affect the action of the compiler.

FILE SECTION

The File Section contains a description of all externally stored data (FD), and a description of each sort-file (SD) used in the program.

The File Section must begin with the header FILE SECTION followed by a period. The File Section contains file description entries and sort-file description entries, each one followed by its associated record description entry (or entries).

General Format
<pre> <u>FILE SECTION.</u> {file description entry {record description entry} ...}...</pre>

File Description Entry

In a COBOL program, the File Description Entries (FD and SD) represent the highest level of organization in the File Section. The File Description entry provides information about the physical structure and identification of a file, and gives the record-name(s) associated with that file.

For a complete discussion of the sort-file-description entry, see "Sort."

Record Description Entry

The Record Description Entry consists of a set of data description entries which describe the particular record(s) contained within a particular file. For a full discussion of the format and the clauses required within the Record Description entry, see "Data Description."

Working-Storage Section

WORKING-STORAGE SECTION

The Working-Storage Section may contain descriptions of records which are not part of external data files but are developed and processed internally.

The Working-Storage Section must begin with the section header WORKING-STORAGE SECTION followed by a period. The Working-Storage Section contains data description entries for noncontiguous items and record description entries, in that order.

General Format
<u>WORKING-STORAGE SECTION.</u> [data item description entry] ... [record description entry] ...

Data Item Description Entries

Noncontiguous items in Working-Storage that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data item description entry that begins with the special level number 77.

Record Description Entries

Data elements in Working-Storage that bear a definite hierarchical relationship to one another must be grouped into records structured by level number.

LINKAGE SECTION

The Linkage Section describes data made available from another program (see "Subprogram Linkage" in "Procedure Division").

General Format
<p><u>LINKAGE SECTION.</u></p> <p>[data item description entry] ...</p> <p>[record description entry] ...</p>
<p>Data item description entries and record description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved since the data area exists elsewhere. Any data description clause may be used to describe items in the Linkage Section, with one exception: the VALUE clause may not be specified for other than level-88 items. In the Linkage Section, the compiler assumes that each level-01 item starts on a double-word boundary.</p> <p><u>Note:</u> The combined total number of level-77 and level-01 items in the Linkage Section may not exceed 255.</p>

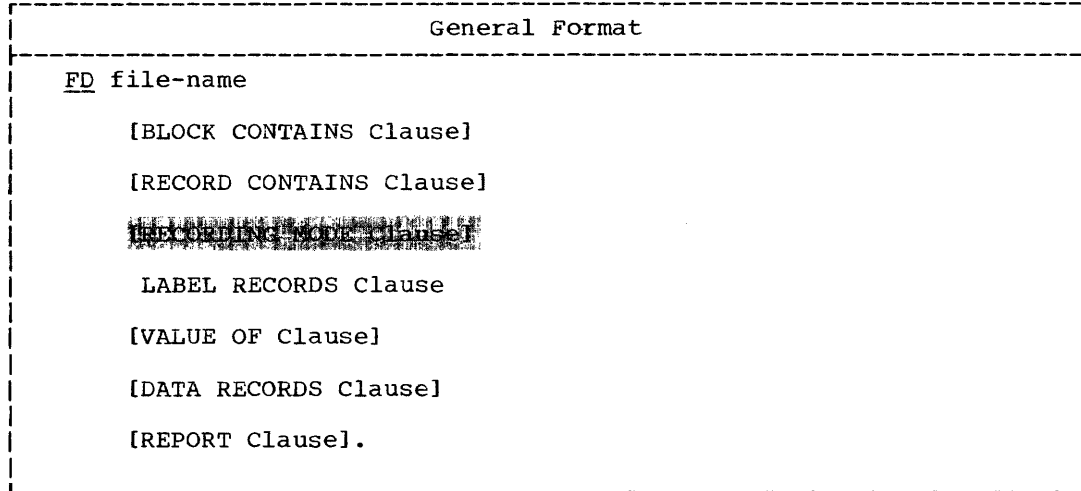
REPORT SECTION

The Report Section contains Report Description entries and report group description entries for every report named in the REPORT clause. The Report Section is discussed in "Report Writer."

FD Entry/BLOCK CONTAINS Clause

FILE DESCRIPTION ENTRY -- DETAILS OF CLAUSES

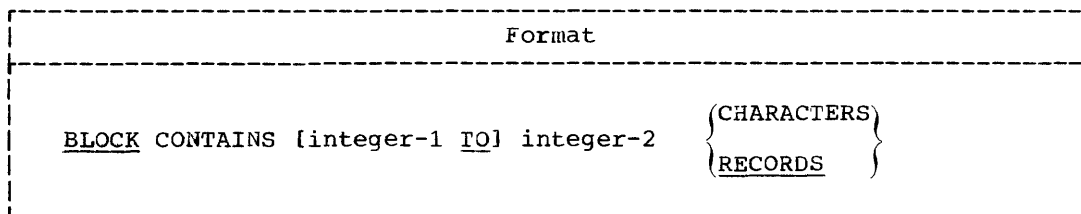
The file description entry consists of level indicator (FD), followed by file-name, followed by a series of independent clauses. The entry itself is terminated by a period.



The level indicator FD identifies the beginning of a file description entry and must precede the file-name. The clauses that follow the name of the file are optional in many cases, and their order of appearance is not significant.

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause is used to specify the size of a physical record.



The BLOCK CONTAINS clause is unnecessary when a physical record contains one and only one complete logical record. In all other instances, this clause is required.

The BLOCK CONTAINS clause need not be specified for:

- direct files with F, U, or V mode records

• direct files when the RECORDING MODE clause is specified for S-mode records

- files containing U-mode records

For these types of files, the compiler accepts the clause and treats it as comments.

The RECORDS option may be used unless one of the following situations exists, in which case the CHARACTERS option should be used:

1. The physical record contains padding (areas not contained in a logical record)
2. Logical records are grouped in such a manner that an inaccurate physical record size would be implied. Such would be the case where the user describes a mode V record of 100 characters, yet each time he writes a block of 4, he writes a 50-character record followed by three 100-character records. Had he used the RECORDS option, the compiler would have calculated the block length as 420.
3. Logical records extend across physical records; that is, recording mode is S (spanned).

When the RECORDS option is used, the compiler assumes that the blocksize provides for integer-2 records of maximum size and then provides additional space for any required control bytes.

When the CHARACTERS option is used, the physical record size is specified in Standard Data Format, i.e., in terms of the number of bytes occupied internally by its characters, regardless of the number of characters used to represent the item within the physical record. The number of bytes occupied internally by a data item is included as part of the discussion of the USAGE clause. Integer-1 and integer-2 must include slack bytes and control bytes contained in the physical record.

When the CHARACTERS option is used, if only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

Integer-1 and integer-2 must be positive integers.

If this clause is omitted, it is assumed that records are not blocked.

When neither the CHARACTERS nor the RECORDS option is specified, the CHARACTERS option is assumed.

Note: ASCII considerations for the BLOCK CONTAINS clause are given in Appendix E.

RECORD CONTAINS Clause

RECORD CONTAINS Clause

The RECORD CONTAINS clause is used to specify the size of a file's data records.

```
-----  
Format  
-----  
RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS  
-----
```

Since the size of each data record is completely defined within the record description entry, this clause is never required. When the clause is specified, the following notes apply:

1. If both integer-1 and integer-2 are shown, they refer to the number of characters in the smallest data record and the number in the largest data record, respectively.
2. Integer-2 should not be used by itself unless all the data records in the file have the same size. In this case, integer-2 represents the exact number of characters in the data record.
3. The size of the record must be specified in Standard Data Format, i.e., in terms of the number of bytes occupied internally by its characters, regardless of the number of characters used to represent the item within the record. The number of bytes occupied internally by a data item is discussed in the description of the USAGE clause. The size of a record is determined according to the rules for obtaining the size of a group item.

Normally, whether this clause is specified or omitted, the record lengths are determined by the compiler from the record descriptions. When one or more of the data item description entries within a record contains an OCCURS clause with the DEPENDING ON option, the compiler uses the maximum value of the variable to calculate the record length.

However, if more than one entry in a given record description contains an OCCURS clause with the DEPENDING ON option, and the maximum values of the variables in these OCCURS clauses do not occur simultaneously, integer-2, as specified by the user, may indicate a maximum record size other than the size calculated by the compiler from the maximum values of the OCCURS clause variables. In this case, the user-specified value of integer-2 determines the amount of storage set aside to contain the data record.

For example, in a school whose total enrollment is 500, an unblocked file of collective attendance records is being created, each record of which is described as follows:

```
01 ATTENDANCE-RECORD.  
02 DATE PICTURE X(6).  
02 NUMBER-ABSENT PICTURE S999 USAGE IS COMP SYNC.  
02 NUMBER-PRESENT PICTURE S999 USAGE IS COMP SYNC.  
02 NAMES-OF-ABSENT OCCURS 0 TO 500 TIMES DEPENDING ON  
    NUMBER-ABSENT PICTURE A(20).  
02 NAMES-OF-PRESENT OCCURS 0 TO 500 TIMES DEPENDING ON  
    NUMBER-PRESENT PICTURE A(20).
```

The programmer can save storage by taking advantage of the fact that NUMBER-ABSENT plus NUMBER-PRESENT will never exceed the school's total enrollment. Unless the programmer writes RECORD CONTAINS 10,010 CHARACTERS in the FD entry for the file, the compiler calculates the record size to be almost twice as large.

Recording Mode

When the RECORDING MODE clause is not used to specify the recording mode of the records in the file, the COBOL compiler scans each record description to determine it. The recording mode may be F (fixed), U (undefined), V (variable), or S (spanned).

Recording Mode F -- All of the records in a file are the same length and each is wholly contained in one block. Blocks may contain more than one record, and there is usually a fixed number of records per block. In this mode, there are no record-length or block-descriptor fields.

Recording Mode U -- The records may be either fixed or variable in length. However, there is only one record per block. There are no record-length or block-descriptor fields.

Recording Mode V -- The records may be either fixed or variable in length, and each must be wholly contained in one block. Blocks may contain more than one record. Each data record includes a record-length field and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

Recording Mode S -- The records may be either fixed or variable in length and may be larger than a block. If a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block (or blocks if required). Only complete records are made available to the user. Each segment of a record in a block, even if it is the entire record, includes a segment-descriptor field, and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

For standard sequential files, the compiler determines the recording mode for a given file to be:

- F if all the records are defined as being the same size and the size is smaller than or equal to the block size
- V if the records are defined as variable in size, or if the RECORD CONTAINS clause specifies variable size records and the longest record is less than or equal to the maximum block size
- S if the maximum block size is smaller than the largest record size

For direct files, the compiler determines the recording mode for a given file to be:

- F if all the records are defined as being the same size, and the size is smaller than or equal to the block size
- U if the records are defined as being variable in size, or if the RECORD CONTAINS clause specifies variable size records and the longest record is less than or equal to the maximum block size

RECORDING MODE Clause

S if the maximum block size is smaller than the largest record size

Files assigned to the card reader and files with indexed organization must be F mode (fixed format).

Note: ASCII considerations for compiler calculation of recording mode are given in Appendix E.

RECORDING MODE Clause

The RECORDING MODE clause is used to specify the format of the logical records in the file.

Format
RECORDING MODE IS mode

Mode may be specified as F, V, U, or S. If this clause is not specified, the recording mode is determined as described in "Recording Mode."

The F mode (fixed-length format) may be specified when all the logical records in a file are the same length and each is wholly contained in one physical block. This implies that no OCCURS clause with the DEPENDING ON option is associated with an entry in any record description for the file. If more than one record description entry is given following the FD entry, all record lengths calculated from the record descriptions must be equal. Files assigned to the card reader and files with indexed organization must be in F mode.

The V mode (variable-length format) may be specified for any combination of record descriptions if each record is wholly contained within one physical block. A mode V record is preceded by a control field containing the length of the logical record. Blocks of variable-length records include a block-descriptor control field. V mode may be specified only for standard sequential files.

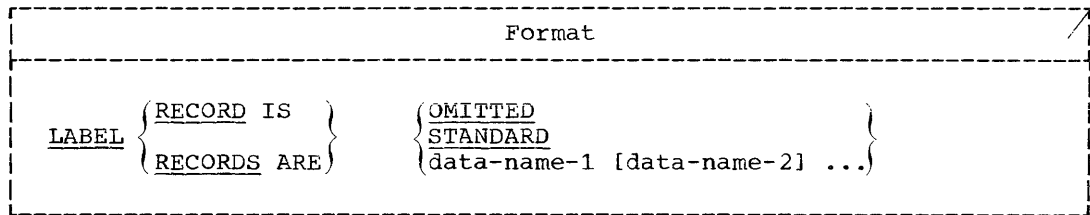
The U mode (undefined format) may be specified for any combination of record descriptions if each record is wholly contained within one physical block. It is comparable to V mode with the exception that U mode records are not blocked and have no preceding control field. U mode may be specified only for direct files or standard sequential files.

The S mode (spanned format) may be specified for any combination of record descriptions. A record that cannot fit into the remaining space in a block appears as multiple segments, one segment per block. A record that can be completely contained in a block appears as a single segment. An S-mode segment is preceded by a control field containing the length of the segment and indicating whether it is the first and/or last or an intermediate segment. Blocks of S-mode segments include a block-descriptor control field. S mode may be specified for standard sequential or direct files.

Note: ASCII considerations for the RECORDING MODE clause are given in Appendix E.

LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether labels are present, and if present, identifies the labels.



The LABEL RECORDS clause is required in every FD.

The OMITTED option specifies either that no explicit labels exist for the file or that the existing labels are nonstandard and the user does not want them to be processed by a label declarative (i.e., they will be processed as data records). The OMITTED option must be specified for files assigned to unit record devices. It may be specified for files assigned to magnetic tape units. Use of the OMITTED option does not result in automatic bypassing of nonstandard labels on input. It is the user's responsibility either to process or to bypass nonstandard labels on input and create them on output.

STANDARD specifies that labels exist for the file and the labels conform to system specification. The system will bypass user labels appearing in the file if the STANDARD option is specified.

The STANDARD option must be specified for files with indexed organization.

In the discussion that follows, all references to data-name-1 apply equally to data-name-2.

The data-name-1 option indicates either the presence of user labels in addition to standard labels, or the presence of nonstandard labels. data-name-1 specifies the name of a user label record. data-name-1 must appear as the subject of a record description entry associated with the file, and must not appear as an operand of the DATA RECORDS clause for the file.

If user labels are to be processed, data-name-1 may be specified for direct files, or for standard sequential files with the exception of files assigned to unit-record devices.

A user label is 80 characters in length. A user header label must have UHL in character positions 1 through 3. A user trailer label must have UTL in character positions 1 through 3. Both header and trailer labels may be grouped and each label must show the relative position (1, 2, ...) of the label within the user label group, in character position 4. The remaining 76 characters are formatted according to the user's choice. User header labels follow standard beginning file labels but precede the first data record; user trailer labels follow standard closing file labels.

If nonstandard labels are to be processed, data-name-1 may be specified only for standard sequential files, with the exception of files assigned to unit-record devices. The length of a nonstandard label may not exceed 4,095 character positions.

VALUE OF/DATA RECORDS Clauses

All Procedure Division references to data-name-1, or to any item subordinate to data-name-1, must appear within label processing declaratives.

Note: ASCII considerations for the LABEL RECORDS clause are given in Appendix E.

VALUE OF Clause

The VALUE OF clause particularizes the description of an item in the label records associated with a file, and serves only as documentation.

Format	
<u>VALUE OF</u> data-name-1 IS	{ data-name-2 literal-1 }
[data-name-3 IS	{ data-name-4 literal-2 }] ...

To specify the required values of identifying data items in the label records for the file, the programmer must use the VALUE OF clause.

However, this compiler treats the VALUE OF clause as comments, since for standard labels this function is performed by the system through the TLBL or DLBL control statement as described in the Programmer's Guides (as cited in "Preface"), and through the Label Declarative procedures for user standard labels and nonstandard labels.

DATA RECORDS Clause

The DATA RECORDS clause serves only as documentation, and identifies the records in the file by name.

Format	
<u>DATA</u> { <u>RECORD IS</u> <u>RECORDS ARE</u> }	data-name-1 [data-name-2] ...

The presence of more than one data-name indicates that the file contains more than one type of data record. That is, two or more record descriptions for a given file occupy the same storage area. These records need not have the same description. The order in which the data-names are listed is not significant.

REPORT Clause

Data-name-1, data-name-2, etc., are the names of data records and each must be preceded in its record description entry by the level number 01.

This clause is never required.

REPORT Clause

The REPORT clause is used in conjunction with the Report Writer feature. A complete description of the REPORT clause can be found in "Report Writer."

DATA DESCRIPTION

In COBOL, the terms used in connection with data description are:

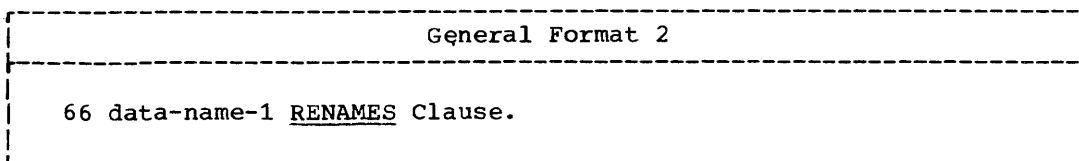
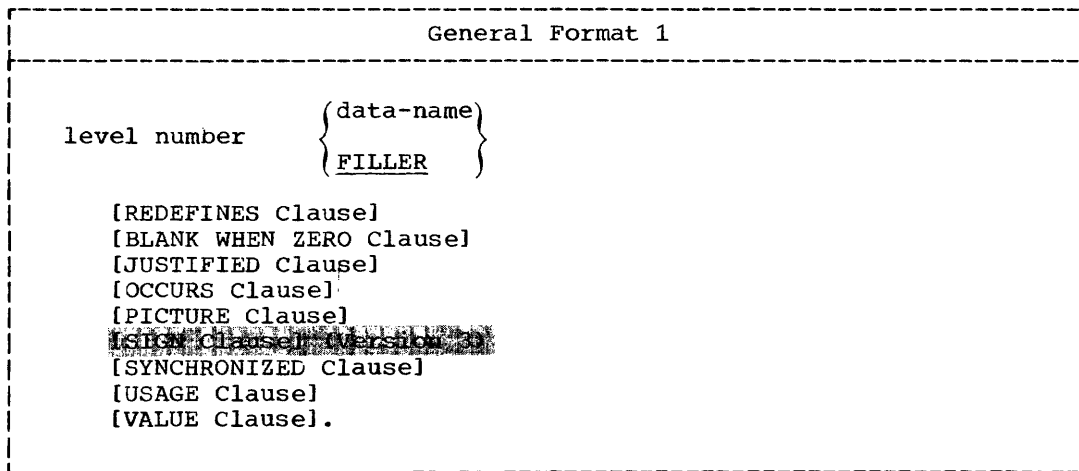
Data Description Entry -- the clause, or clauses, that specify the characteristics of any particular noncontiguous data item, or of any data item that is a portion of a record. The data description entry consists of a level number, a data-name (or condition-name), plus any associated data description clauses.

Data Item Description Entry -- a data description entry that defines a noncontiguous data item. It consists of a level number (77), a data-name plus any associated data description entries. Data item description entries are valid in the Working-Storage Section, and in the Linkage Section.

Record Description Entry -- the term used in connection with a record. It consists of a hierarchy of data description entries. Record description entries are valid in the File, Working-Storage Sections.

Note: For the 3881 optical mark reader, the first 6 bytes of the record description entry should be described as a FILLER item; these 6 bytes are reserved for control information and are not available to the COBOL program.

The maximum length for a data description entry is 32,767 bytes, except for a fixed-length Working-Storage Section group item, which may be as long as 131,071 bytes.



General Format 3

88 condition-name VALUE Clause.

General Format 1 is used for record description entries in the File, Working-Storage, and Linkage Sections and for data item description entries in the Working-Storage and Linkage Sections. The following rules apply:

1. Level number may be any number from 1 through 49 for record description entries, or 77 for data item description entries.
2. The clauses may be written in any order, with one exception: the REDEFINES clause, when used, must immediately follow the data-name.
3. The PICTURE clause must be specified for every elementary item, with the exception of index data items and internal floating point items. Index data items are described in "Table Handling."
4. Each entry must be terminated by a period.
5. Semicolons or commas may be used as separators between clauses.

General Format 2 is used for the purpose of regrouping data items. The following rules apply:

1. A level-66 entry cannot rename another level-66 entry, nor can it rename a level-77, level-88, or level-01 entry.
2. All level-66 entries associated with a given logical record must immediately follow the last data description entry in the record.
3. The entry must be terminated by a period.

The RENAMES clause is discussed in detail later in this chapter.

General Format 3 is used to describe entries that specify condition-names to be associated with particular values of a conditional variable. A condition-name is a name assigned by the user to a specific value that a data item may assume during object program execution. The following rules apply:

1. The condition-name entries for a particular conditional variable must immediately follow the conditional variable.
2. A condition-name can be associated with any elementary data description entry except another condition-name, or an index data item.
3. A condition-name can be associated with a group item data description entry. In this case:
 - The condition value must be specified as a nonnumeric literal or figurative constant.
 - The size of the condition value must not exceed the sum of the sizes specified by the pictures in all the elementary items within the group.

Data Description -- General Formats

- No element within the group may contain a JUSTIFIED or SYNCHRONIZED clause.
 - No USAGE other than USAGE IS DISPLAY may be specified within the group.
4. The specification of a condition-name at the group level does not restrict the specification of condition-names at levels subordinate to that group.
 5. The relation test implied by the definition of a condition-name at the group level is performed in accordance with the rules for comparison of nonnumeric operands, regardless of the nature of elementary items within the group.
 6. Each entry must be terminated by a period.

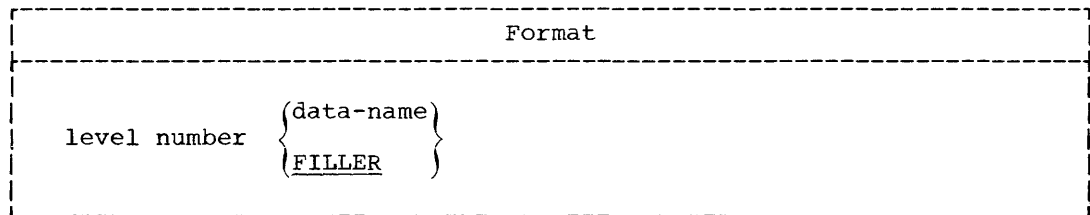
Examples of both group and elementary condition-name entries are given in the description of the VALUE clause.

DATA DESCRIPTION ENTRY -- DETAILS OF CLAUSES

The data description entry consists of a level number, followed by a data-name, followed by a series of independent clauses. The clauses may be written in any order, with one exception: the REDEFINES clause, when used, must immediately follow the data-name. The entry must be terminated by a period.

Data-name or FILLER Clause

A data-name specifies the name of the data being described. The word FILLER specifies an elementary ~~item or group~~ item of the logical record that is never referred to and therefore need not be named.



In the Working-Storage, ~~Package~~, or File Sections, a data-name or the key word FILLER must be the first word following the level number in each data description entry.

A data-name is a name assigned by the user to identify a data item used in a program. A data-name refers to a kind of data, not to a particular value; the item referred to may assume a number of different values during the course of a program.

The key word FILLER is used to specify an elementary ~~item or group~~ item that is never referred to in the program, and therefore need not be named. Under no circumstances may a FILLER item be referred to directly. In a MOVE, ADD, or SUBTRACT statement with the CORRESPONDING option, FILLER items are ignored.

Note: Level-77 and level-01 entries in the Working-Storage ~~Package~~ Section must be given unique data-names, since neither can be qualified. Subordinate data-names, if they can be qualified, need not be unique.

REDEFINES Clause

REDEFINES Clause

The REDEFINES clause allows the same computer storage area to contain different data items or provides an alternative grouping or description of the same data. That is, the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

```
                                Format
-----
level number data-name-1 REDEFINES data-name-2
```

The level numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88. Data-name-2 is the name associated with the previous data description entry. Data-name-1 is an alternate name for the same area. When written, the REDEFINES clause must be the first clause following data-name-1.

The REDEFINES clause must not be used in level-01 entries in the File Section. Implicit redefinition is provided when more than one level-01 entry follows a file description entry.

Redefinition starts at data-name-2 and ends when a level number less than or equal to that of data-name-2 is encountered. Between the data descriptions of data-name-2 and data-name-1, there may be no entries having lower level numbers (numerically) than the level number of data-name-2 and data-name-1. Example:

```
02  A.
    03  A-1      PICTURE X.
    03  A-2      PICTURE XXX.
    03  A-3      PICTURE 99.
02  B REDEFINES A PICTURE X(6).
```

In this case, B is data-name-1, and A is data-name-2. When B redefines A, the redefinition includes all of the items subordinate to A (A-1, A-2, and A-3).

The data description entry for data-name-2 cannot contain an OCCURS clause, nor can data-name-2 be subordinate to an entry which contains an OCCURS clause. An item subordinate to data-name-2 may contain an OCCURS clause without the DEPENDING ON option. Data-name-1 or any items subordinate to data-name-1 may contain an OCCURS clause without the DEPENDING ON option. Neither data-name-2 nor data-name-1 nor any of their subordinate items may contain an OCCURS clause with the DEPENDING ON option.

When data-name-1 has a level number other than 01, it must specify a storage area of the same size as data-name-2.

However, this compiler will allow the size of the redefined data-name-1 to be less than the size of the redefining data-name-2.

If data-name-1 contains an OCCURS clause, its size is computed by multiplying the length of one occurrence by the number of occurrences.

Note: In the discussion that follows, the term "computational" refers to COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items.

When the SYNCHRONIZED clause is specified for an item that also contains a REDEFINES clause, the data item that is redefined must have the proper boundary alignment for the data item that REDEFINES it. For example, if the programmer writes:

```
02 A          PICTURE X(4).
02 B REDEFINES A PICTURE S9(9) COMP SYNC.
```

he must ensure that A begins on a fullword boundary.

When the SYNCHRONIZED clause is specified for a computational item that is subordinate to an item that contains a REDEFINES clause, the computational item must not require the addition of slack bytes.

Except for condition-name entries, the entries giving the new description of the storage area must not contain any VALUE clauses.

The entries giving the new description of the storage area must follow the entries describing the area being redefined, without intervening entries that define new storage areas. Multiple redefinitions of the same storage area should all use the data-name of the entry that originally defined the area. However, this compiler will accept as valid the data-name of the preceding entry when multiple redefinition is used. For example, both of the following are valid uses of the REDEFINES clause:

```
02 A          PICTURE 9999.
02 B REDEFINES A PICTURE 9V999.
02 C REDEFINES A PICTURE 99V99.

02 A          PICTURE 9999.
02 B REDEFINES A PICTURE 9V999.
02 C REDEFINES B PICTURE 99V99.
```

Data items within an area can be redefined without their lengths being changed; the following statements result in the storage layout shown in Figure 12.

```
02 NAME-2.
03 SALARY    PICTURE XXX.
03 SO-SEC-NO PICTURE X(9).
03 MONTH     PICTURE XX.
02 NAME-1 REDEFINES NAME-2.
03 WAGE      PICTURE XXX.
03 MAN-NO    PICTURE X(9).
03 YEAR      PICTURE XX.
```

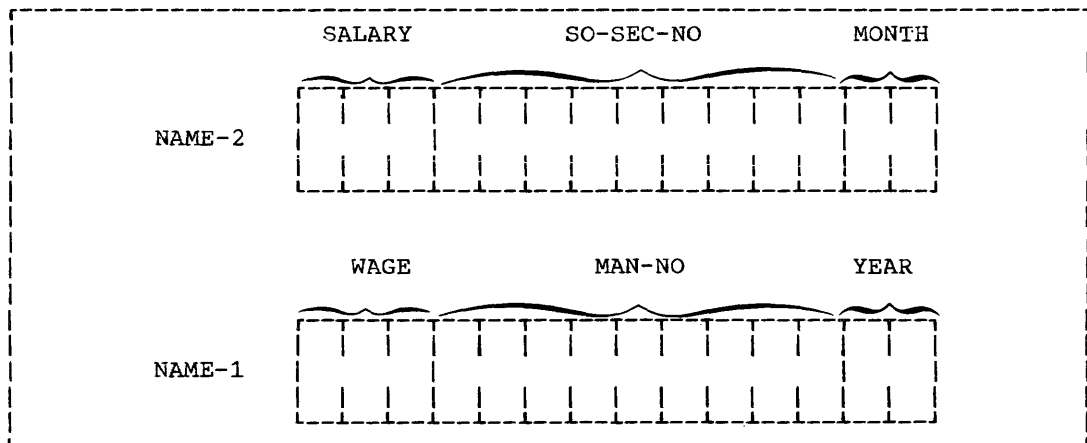


Figure 12. Areas REDEFINED without Changes in Length

REDEFINES Clause

Data items can also be rearranged within an area; the following statements result in the storage layout shown in Figure 13.

```

02 NAME-2.
03 SALARY      PICTURE XXX.
03 SO-SEC-NO  PICTURE X(9).
03 MONTH      PICTURE XX.
02 NAME-1 REDEFINES NAME-2.
03 MAN-NO     PICTURE X(6).
03 WAGE       PICTURE 999V9999.
03 YEAR       PICTURE XX.
  
```

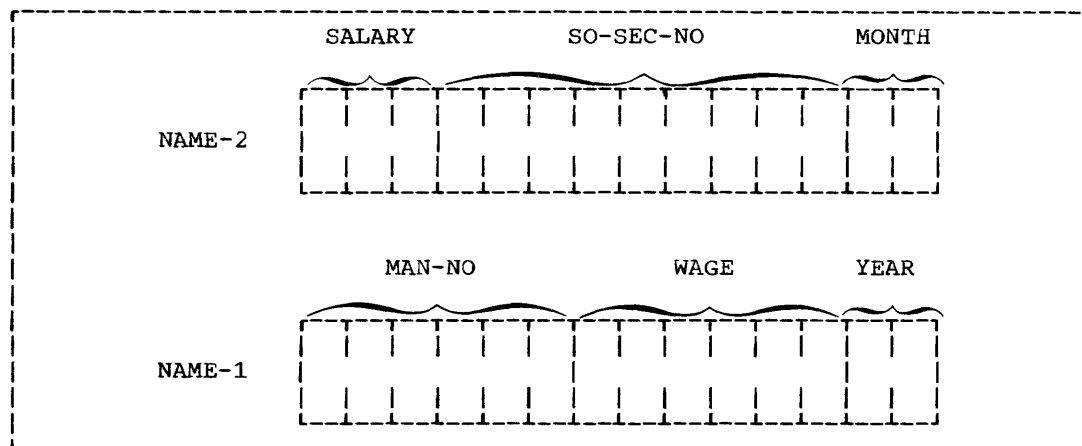


Figure 13. Areas REDEFINED and Rearranged

When an area is redefined, all descriptions of the area remain in effect. Thus, if B and C are two separate items that share the same storage area due to redefinition, the procedure statements MOVE X TO B or MOVE Y TO C could be executed at any point in the program. In the first case, B would assume the value of X and take the form specified by the description of B. In the second case, the same physical area would receive Y according to the description of C. It should be noted, however, that if both of the foregoing statements are executed successively in the order specified, the value Y will overlay the value X. However, redefinition in itself does not cause any data to be erased and does not supersede a previous description.

The usage of data items within an area can be redefined.

Altering the USAGE of an area through redefinition does not cause any change in existing data. Consider the example:

```

02 B          PICTURE 99 USAGE DISPLAY VALUE IS 8.
02 C REDEFINES B PICTURE S99 USAGE COMPUTATIONAL.
02 A          PICTURE S9999 USAGE COMPUTATIONAL.
  
```

Assuming that B is on a halfword boundary, the bit configuration of the value 8 is 1111 0000 1111 1000, because B is a DISPLAY item. Redefining B does not change its appearance in storage. Therefore, a great difference results from the two statements ADD B TO A and ADD C TO A. In the former case, the value 8 is added to A, because B is a display item. In the latter case, the value -3,848 is added to A, because C is a binary item (USAGE IS COMPUTATIONAL), and the bit configuration appears as a negative number.

Moving a data item to a second data item that redefines the first one (for example, MOVE B TO C when C redefines B), may produce results that are not those expected by the programmer. The same is true of the reverse (MOVE B TO C when B redefines C).

A REDEFINES clause may be specified for an item within the scope of an area being redefined, that is, an item subordinate to a redefined item. The following example would thus be a valid use of the REDEFINES clause:

```

02  REGULAR-EMPLOYEE.
    03  LOCATION                PICTURE A(8).
    03  STATUS                   PICTURE X(4).
    03  SEMI-MONTHLY-PAY        PICTURE 9999V99.
    03  WEEKLY-PAY REDEFINES SEMI-MONTHLY-PAY PICTURE 999V999.
02  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
    03  LOCATION                PICTURE A(8).
    03  FILLER                   PICTURE X(6).
    03  HOURLY-PAY              PICTURE 99V99.

```

REDEFINES clauses may also be specified for items subordinate to items containing REDEFINES clauses. For example:

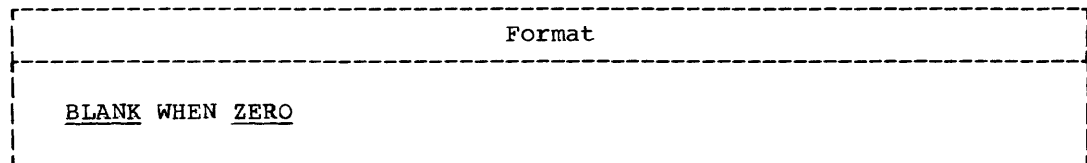
```

02  REGULAR-EMPLOYEE.
    03  LOCATION                PICTURE A(8).
    03  STATUS                   PICTURE X(4).
    03  SEMI-MONTHLY-PAY        PICTURE 999V999.
02  TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
    03  LOCATION                PICTURE A(8).
    03  FILLER                   PICTURE X(6).
    03  HOURLY-PAY              PICTURE 99V99.
    03  CODE-H REDEFINES HOURLY-PAY PICTURE 9999.

```

BLANK WHEN ZERO Clause

This clause specifies that an item is to be set to blanks whenever its value is zero.



When the BLANK WHEN ZERO clause is used, the item will contain only blanks if the value of the item is zero.

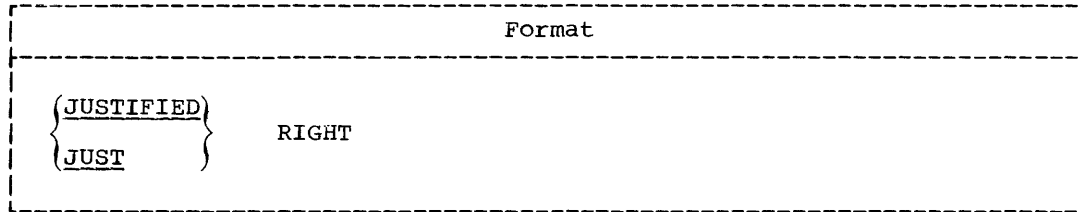
The BLANK WHEN ZERO clause may be specified only at the elementary level for numeric edited or numeric items. When this clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

This clause may not be specified for level-66 and level-88 data items.

JUSTIFIED Clause

JUSTIFIED Clause

The JUSTIFIED clause is used to override normal positioning of data within a receiving alphabetic or alphanumeric data item.



Normally, the rule for positioning data within a receiving alphanumeric or alphabetic data item is:

- The data is aligned in the receiving field, beginning at the leftmost character position within the receiving field. Unused character positions to the right are filled with spaces. If truncation occurs, it will be at the right.

The JUSTIFIED clause affects the positioning of data in the receiving field as follows:

- When the receiving data item is described with the JUSTIFIED clause and the data item sent is larger than the receiving data item, the leftmost characters are truncated.
- When the receiving data item is described with the JUSTIFIED clause and is larger than the data item sent, the data is aligned at the rightmost character position in the data item. Unused character positions to the left are filled with spaces.

The JUSTIFIED clause may only be specified for elementary items.

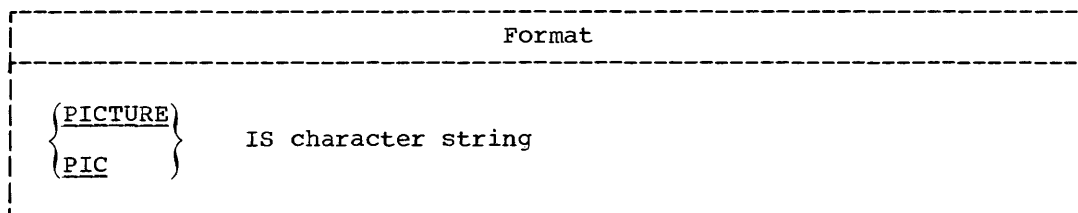
This clause must not be specified for level-66 or level-88 data items.

OCCURS Clause

The OCCURS clause is used to define tables and other homogeneous sets of data, whose elements can be referred to by subscripting or indexing. The OCCURS clause is described in "Table Handling."

PICTURE Clause

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.



The PICTURE clause can be used only at the elementary level.

The character string consists of certain allowable combinations of characters in the COBOL character set. The maximum number of characters allowed in the character string is 30. The allowable combinations determine the category of the elementary item.

There are five categories of data that can be described with a PICTURE clause. They are:

1. Alphabetic
2. Numeric
3. Alphanumeric
4. Alphanumeric edited
5. Numeric edited

The Three Classes of Data

The five categories of data items are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the class and the category are synonymous. The alphanumeric class includes the categories of alphanumeric (without editing), alphanumeric edited, and numeric edited.

Every elementary item belongs to one of the three classes and to one of the five categories. The class of a group item is treated at object time as alphanumeric regardless of the class of the elementary items subordinate to that group item.

Figure 14 shows the relationship of the class and category for elementary and group data items.

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Alphanumeric Alphanumeric Edited Numeric Edited
Group	Alphanumeric	Alphabetic Numeric Alphanumeric Alphanumeric Edited Numeric Edited

Figure 14. Class and Category of Elementary and Group Data Items

PICTURE Clause

Character String and Item Size

In the processing of data through COBOL statements, the size of an elementary item is determined through the number of character positions specified in its PICTURE character string. In core storage, however, the size is determined by the actual number of bytes the item occupies, as determined by its PICTURE character string, and also by its USAGE (see "USAGE Clause").

Normally, when an arithmetic item is moved from a longer field into a shorter one, this compiler will truncate the data to the number of characters represented in the PICTURE character string of the shorter item.

For example, if a sending field with PICTURE S99999, and containing the value +12345, is moved to a COMPUTATIONAL receiving field with PICTURE S99, the data is truncated to +45.

As a compile time option, however, this compiler may be instructed, in such an operation, to truncate only such digits as would overflow the receiving field. In this option is used, the result of the move in the foregoing example is +2345, since a COMPUTATIONAL item two bytes in length can contain up to four decimal digits of data. Note that care must be used when using this option, since there are times when the data

Repetition of Symbols

An integer which is enclosed in parentheses following one of the symbols

A , X 9 P Z * B 0 + - \$

indicates the number of consecutive occurrences of the symbol. For example, if the programmer writes

A(40)

the four characters (40) indicate forty consecutive appearances of the symbol A. The number within parentheses may not exceed 32,767.

Note: The following symbols may appear only once in a given PICTURE clause:

S V . CR DB

Symbols Used in the PICTURE Clause

The functions of the symbols used to describe an elementary item are:

- A Each A in the character string represents a character position that can contain only a letter of the alphabet or a space.
- B Each B in the character string represents a character position into which the space character will be inserted.

The E in the character string represents the exponent in an external floating-point item. The E occupies one byte of storage, and is counted in determining the size of the elementary item. The E is included in any representation upon external media.

P The P indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or in items that appear as operands in arithmetic statements.

The scaling position character P may appear only to the left or right of the other characters in the string as a continuous string of P's within a PICTURE description. The sign character S and the assumed decimal point V are the only characters which may appear to the left of a leftmost string of P's. Since the scaling position character P implies an assumed decimal point (to the left of the P's if the P's are leftmost PICTURE characters and to the right of the P's if the P's are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description.

S The symbol S is used in a PICTURE character string to indicate the presence (but not the representation nor, necessarily, the position) of an operational sign, and must be written as the leftmost character in the PICTURE string. An operational sign indicates whether the value of an item involved in an operation is positive or negative. The symbol S is not counted in determining the size of the elementary item, unless an associated SIGN clause specifies the SEPARATE CHARACTER option.

V The V is used in a character string to indicate the location of the assumed decimal point and may appear only once in a character string. The V does not represent a character position and, therefore, is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant.

X Each X in the character string represents a character position which may contain any allowable character from the EBCDIC set.

Z Each Z in the character string represents a leading numeric character position; when that position contains a zero, the zero is replaced by a space character. Each Z is counted in the size of the item.

9 Each 9 in the character string represents a character position that contains a numeral and is counted in the size of the item.

0 Each zero in the character string represents a character position into which the numeral zero will be inserted. The 0 is counted in the size of the item.

,

Each comma in the character string represents a character position into which a comma will be inserted. This character is counted in the size of the item. The comma insertion character cannot be the last character in the PICTURE character string.

.

When a period appears in the character string, it is an editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period will be inserted. This character is counted in the size of the item. The period insertion character cannot be the last character in the PICTURE character string.

Note: For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange, the rules for the period apply to the comma and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

PICTURE Clause

- { + } These symbols are used as editing sign control symbols. When used, each represents the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in one character string. Each character used in the symbol is counted in determining the size of the data item.
- { - }
- { CR }
- { DB }
- * Each asterisk (check protect symbol) in the character string represents a leading numeric character position into which an asterisk will be placed when that position contains a zero. Each * is counted in the size of the item.
- \$ The currency symbol in the character string represents a character position into which a currency symbol is to be placed. The currency symbol in a character string is represented either by the symbol \$ or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Figure 15 shows the order of precedence of the symbols used in the PICTURE clause.

The Five Categories of Data

The following is a detailed description of the allowable combinations of characters for each category of data.

ALPHABETIC ITEMS: An alphabetic item is one whose PICTURE character string contains only the symbol A. Its contents, when represented in Standard Data Format, must be any combination of the 26 letters of the Roman alphabet and the space from the COBOL character set. Each alphabetic character is stored in a separate byte.

If a VALUE clause is specified for an alphabetic item, the literal must be nonnumeric.

ALPHANUMERIC ITEMS: An alphanumeric item is one whose PICTURE character string is restricted to combinations of the symbols A, X, and 9. The item is treated as if the character string contained all X's. Its contents, when represented in Standard Data Format, are allowable characters from the EBCDIC set.

A PICTURE character string which contains all A's or all 9's does not define an alphanumeric item.

If a VALUE clause is specified for an alphanumeric item, the literal must be nonnumeric.

FIRST SYMBOL \ SECOND SYMBOL	FIRST SYMBOL	NON-FLOATING INSERTION SYMBOLS					FLOATING INSERTION SYMBOLS					OTHER SYMBOLS							
	B 0 , . { + } { - } { CR } { DB } cs ¹ { Z } { * } { + } { - } cs ¹ cs ¹ 9 A X S V P P E ²	B	0	,	.	{ + } { - }	{ CR } { DB }	cs ¹	{ Z } { * }	{ + } { - }	cs ¹	cs ¹	9	A	X	S	V	P	P
NON-FLOATING INSERTION SYMBOLS	B	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	0	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	,	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	.	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	{ + or - }																		X
	{ + or - }	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	{ CR or DB }	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
FLOATING INSERTION SYMBOLS	cs ¹					X													
	{ Z or * }	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	{ Z or * }	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	{ + or - }	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	{ + or - }	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	cs ¹	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
OTHER SYMBOLS	cs ¹	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	9	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	A X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	S																		
	V	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	P	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X
	P					X		X	X	X	X	X	X	X	X	X	X	X	X

¹cs is the abbreviation for the currency symbol.

²See the description of external floating-point items for the specific combination of symbols that is valid.

At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, -, or cs must be present in a PICTURE string.

An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.

Non-floating insertion symbols + and -, floating insertion symbols Z, *, +, -, and cs, and other symbol P appear twice in the above PICTURE character precedence table. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.

Braces ({}) indicate items that are mutually exclusive

Figure 15. Precedence of Symbols Used in the PICTURE Clause

PICTURE Clause

NUMERIC ITEMS: There are two types of numeric items: fixed-point items and floating-point items.

Fixed-Point Numeric Items: There are three types of fixed-point numeric items: external decimal, binary, and internal decimal. See the discussion of the USAGE clause for details concerning each.

The PICTURE of a fixed-point numeric item may contain a valid combination of the following symbols:

9 V P S

Examples of fixed-point numeric items:

<u>PICTURE</u>	<u>Valid Range of Values</u>
9999	0 through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	0 through .000999
S999PPP	-1000 through -999000 and +1000 through +999000 or zero

The maximum size of a fixed-point numeric item is 18 digits.

The contents of a fixed-point numeric item, when represented in Standard Data Format, must be a combination of the Arabic numerals 0 through 9; the item may contain an operational sign. If the PICTURE contains an S, the contents of the item are treated as positive or negative values, depending on the operational sign; if the PICTURE does not contain an S, the contents of the item are treated as absolute values.

Note: ASCII considerations for the PICTURE clause are given in Appendix E.

Floating-Point Numeric Items: These items define data whose potential range of value is too great for fixed-point presentation. The magnitude of the number represented by a floating-point item must be greater than 5.4×10^{-79} but must not exceed $.72 \times 10^{76}$.

There are two types of floating-point items: internal floating-point and external floating-point. See the discussion of the USAGE clause for details concerning each.

No PICTURE clause may be associated with an internal floating-point item.

If a VALUE clause is specified for an elementary numeric item, the literal must be numeric. If a VALUE clause is specified for a group item consisting of elementary numeric items, the group is considered alphanumeric, and the literal must therefore be nonnumeric.

An external floating-point item has a PICTURE character string in the following form:

$$\left\{ \begin{array}{c} + \\ - \end{array} \right\} \text{ mantissa } E \left\{ \begin{array}{c} + \\ - \end{array} \right\} \text{ exponent}$$

where each element of the string is composed according to the following rules:

{+} A plus sign or a minus sign must immediately precede both the mantissa and the exponent in the PICTURE character string.
 {-}

+ indicates that a plus sign in the data represents positive values and a minus sign represents negative values.

- indicates that a space character in the data represents positive values and that a minus sign represents negative values.

The plus sign, the space character, and the minus sign occupy a byte of storage, and are included in the printout.

mantissa The mantissa immediately follows the first sign character, and is represented using the following three symbols:

9 Each 9 in the mantissa character string represents a digit position into which a numeric character will be placed. From one to sixteen 9's may be present in the string. Each digit position occupies a byte of storage.

. indicates an actual decimal point. It occupies a byte of storage.

V indicates an assumed decimal point. It does not take up any storage.

One actual or assumed decimal point must be present in the mantissa as a leading, embedded or trailing symbol.

E indicates the exponent, and immediately follows the mantissa. It occupies one byte of storage.

exponent The exponent immediately follows the second sign character. It is represented by two consecutive 9's. Each occupies a byte of storage.

External data must conform to the representation specified in the PICTURE clause.

Examples of external floating-point items:

<u>PICTURE</u>	<u>Format of External Data</u>	<u>Value Expressed</u>
-9V99E-99	540E-79	+5.40 x 10 ⁷⁹
+999.99E+99	+123.45E-14	+123.45 x 10 ¹⁴
-V9(6)E+99	565656E+45	+565656 x 10 ⁴⁵
+9(10)E-99	+7200000000E 76	+72 x 10 ⁷⁶

(Note that any of the above PICTURE representations can express the full range of possible values.)

No VALUE clause may be associated with an external floating-point item.

PICTURE Clause

ALPHANUMERIC EDITED ITEMS: An alphanumeric edited item is one whose PICTURE character string is restricted to certain combinations of the following symbols:

A X 9 B 0

To qualify as an alphanumeric edited item, one of the following conditions must be true:

1. The character string must contain at least one B and at least one X.
2. The character string must contain at least one 0 and at least one X.
3. The character string must contain at least one 0 (zero) and at least one A. Its contents, when represented in Standard Data Format, are allowable characters chosen from the EBCDIC set.

USAGE IS DISPLAY is used in conjunction with alphanumeric edited items.

If a VALUE clause is specified for an alphanumeric edited item, the literal must be nonnumeric. The literal is treated exactly as specified; no editing is performed.

Editing Rules: Alphanumeric edited items are subject to only one type of editing: simple insertion using the symbols 0 and B.

Examples of alphanumeric edited items:

<u>PICTURE</u>	<u>Value of Data</u>	<u>Edited Result</u>
000X(12)	ALPHANUMER01	000ALPHANUMER01
BBBX(12)	ALPHANUMER01	ALPHANUMER01
000A(12)	ALPHABETIC	000ALPHABETIC
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC

NUMERIC EDITED ITEMS: A numeric edited item is one whose PICTURE character string is restricted to certain combinations of the symbols:

B P V Z 0 9 , . * + - CR DB \$

The allowable combinations are determined from the order of precedence of symbols and editing rules.

The maximum number of digit positions that may be represented in the character string is 18.

The contents of the character positions that represent a digit, in Standard Data Format, must be one of the numerals.

USAGE IS DISPLAY is used in conjunction with numeric edited items.

If a VALUE clause is specified for a numeric edited item, the literal must be nonnumeric. The literal is treated exactly as specified; no editing is performed.

The maximum length of a numeric edited item is 127 characters.

Editing Rules: All types of editing are valid for numeric edited items.

Types of Editing

There are two general methods of performing editing in the PICTURE clause: by insertion or by suppression and replacement.

There are four types of insertion editing:

1. simple insertion
2. special insertion
3. fixed insertion
4. floating insertion

There are two types of suppression and replacement editing:

1. zero suppression and replacement with spaces
2. zero suppression and replacement with asterisks

Insertion Editing

Simple insertion editing is performed using the following insertion characters:

, (comma) B (space) 0 (zero)

The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

Examples of simple insertion editing:

<u>PICTURE</u>	<u>Value of Data</u>	<u>Edited Result</u>
99,999	12345	12,345
9,999,000	12345	2,345,000
99B999B000	1234	01 234 000
99B999B000	12345	12 345 000
99BBB999	123456	23 456

Special insertion editing is performed using the period (.) as the insertion character. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character string.

In addition to being an insertion character, the period represents a decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item.

The use of both the assumed decimal point, represented by the symbol V, and the actual decimal point, represented by the period insertion character, in one PICTURE character string is not allowed.

Examples of special insertion editing:

<u>PICTURE</u>	<u>Value of Data</u>	<u>Edited Result</u>
999.99	1.234	001.23
999.99	12.34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

PICTURE Clause

Fixed insertion editing is performed by using the following insertion characters:

currency symbol \$
 editing sign control symbols + - CR DB

Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character string.

Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character string.

- \$ The currency symbol must be the leftmost character position to be counted in the size of the item, unless it is preceded by either a + or a - symbol.
- + or - When either symbol is used, it must represent the leftmost or rightmost character position to be counted in the size of the item.
- CR or DB When either symbol is used, it represents two character positions in determining the size of the item and must represent the rightmost character positions that are counted in the size of the item.

Editing sign control symbols produce results depending upon the value of the data item as shown in Figure 16.

Editing Symbol in PICTURE Character String	Result	
	Data Item Positive or Zero	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Figure 16. Editing Sign Control Symbols and their Results

Examples of fixed insertion editing:

<u>PICTURE</u>	<u>Value of Data</u>	<u>Edited Result</u>
999.99+	+6555.556	555.55+
+9999.99	-5555.555	-5555.55
9999.99-	+1234.56	1234.56
\$999.99	-123.45	\$123.45
-\$999.99	-123.456	-\$123.45
\$9999.99CR	+123.45	\$0123.45
\$9999.99DB	-123.45	\$0123.45DB

Floating insertion editing is indicated in a PICTURE character string by using a string of at least two of the allowable insertion characters \$ + or - to represent the leftmost numeric character positions into which the insertion characters can be floated.

The currency symbol (\$) and the editing sign symbols (+ or -) are mutually exclusive as floating insertion characters in a given PICTURE character string.

Any of the simple insertion characters (, B 0) embedded in the string of floating insertion characters, or to the immediate right of this string, are part of the floating string.

In a PICTURE character string, there are only two ways of representing floating insertion editing:

1. Any or all leading numeric character positions to the left of the decimal point are represented by the insertion character.
2. All of the numeric character positions in the PICTURE character string are represented by the insertion character.

The result of floating insertion editing depends upon the representation in the PICTURE character string:

1. If the insertion characters are only to the left of the decimal point, a single insertion character is placed into the character position immediately preceding the first nonzero digit in the data represented by the insertion symbol string or the decimal point, whichever is farther to the left of the PICTURE character string.
2. If all numeric character positions in the PICTURE character string are represented by the insertion character, the result depends upon the value of the data. If the value is zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion characters are only to the left of the decimal point.

To avoid truncation when using floating insertion editing, the programmer must specify the minimum size of the PICTURE character string for the receiving data item to be:

1. The number of characters in the sending item, plus
2. The number of insertion characters (other than floating insertion characters) being edited into the receiving data item, plus
3. One character for the floating insertion character.

Examples of floating insertion editing:

<u>PICTURE</u>	<u>Value of Data</u>	<u>Edited Result</u>
\$\$\$\$.99	.123	\$.12
\$\$\$\$9.99	.12	\$0.12
\$\$, \$\$\$, 999.99	-1234.56	\$1,234.56
++, +++, 999.99	-123456.789	-123,456.78
\$\$, \$\$\$, \$\$\$, 99CR	-1234567	\$1,234,567.00CR
\$\$, \$\$\$, \$\$\$, 99DB	+1234567	\$1,234,567.00
++, +++, +++, +++)	0000.00	

PICTURE Clause

Zero Suppression and Replacement Editing

Zero suppression and replacement editing means the suppression of leading zeros in numeric character positions and is indicated by the use of the alphabetic character Z or the character * in the PICTURE character string. If Z is used, the replacement character will be the space; if * is used, the replacement character will be *.

The symbols + - * Z and \$ are mutually exclusive as floating replacement characters in a given PICTURE character string.

Each suppression symbol is counted in determining the size of an item.

Zero suppression and replacement editing is indicated in a PICTURE character string by using a string of one or more of either allowable symbol to represent leading numeric character positions, which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string. Simple insertion or fixed insertion editing characters to the left of the string are not included.

In a PICTURE character string, there are only two ways of representing zero suppression:

1. Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols.
2. All of the numeric character positions in the PICTURE character string are represented by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which appears in a character position corresponding to a suppression symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character string are represented by suppression symbols, and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point.

If the value of the data is zero, the entire data item will be spaces if the suppression symbol is Z, or it will be asterisks (except for the actual decimal point) if the suppression symbol is *.

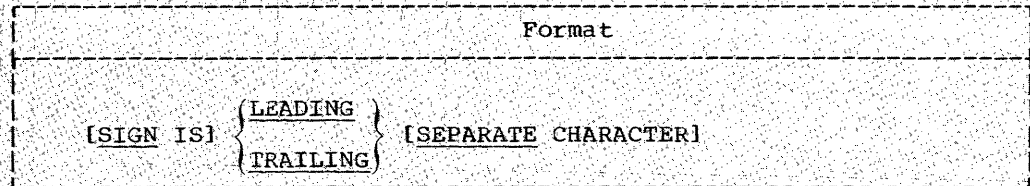
If the value of the data is zero and the asterisk is used as the suppression symbol, zero suppression editing overrides the function of the BLANK WHEN ZERO clause, if specified.

Examples of Zero Suppression and Replacement Editing:

<u>PICTURE</u>	<u>Value of Data</u>	<u>Edited Result</u>
ZZZZ.ZZ	0000.00	
****.**	0000.00	****.**
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	00000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+
*,***.**	-123.45	**123.45-
,,***,***.***	+12345678.9	12,345,678.90+
\$Z,ZZZ,ZZZ.ZZCR	+12345.67	\$ 12,345.67
\$B*,***,***.**BBDB	-12345.67	\$ ***12,345.67 DB

Program Product Information -- Version 3SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric data description entry.



The SIGN clause is required only when an explicit description of the properties of the operational sign is necessary.

The numeric data description entries to which the SIGN clause applies must, explicitly or implicitly, be described as USAGE IS DISPLAY.

Only one SIGN clause may apply to any given numeric data description entry.

The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character S, or for a group item containing at least one such numeric data description entry.

When specified, the SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER option is not specified, then:

- The operational sign is presumed to be associated with the LEADING or TRAILING digit position, whichever is specified, of the elementary numeric data item. (In this instance, specification of SIGN IS TRAILING is the equivalent of the standard action of the compiler.)
- The character S in the PICTURE character string is not counted in determining the size of the item (in terms of Standard Data Format characters).

If the SEPARATE CHARACTER option is specified, then:

- The operational sign is presumed to be the LEADING or TRAILING character position, whichever is specified, of the elementary numeric data item. This character position is not a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of Standard Data Format characters).
- + is the character used for the positive operational sign.
- - is the character used for the negative operational sign.

SYNCHRONIZED Clause

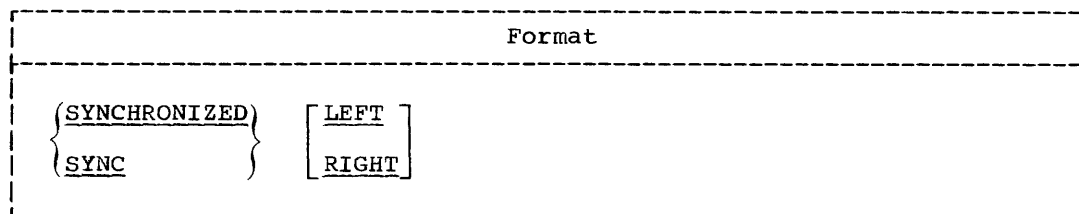
- At object time if one of the characters + or - is not present in the data an error occurs, and the program will terminate abnormally.

Every numeric data description entry whose PICTURE contains the character S is a signed numeric data description entry. If the SIGN clause applies to such an entry, and conversion is necessary for purposes of computation, or for comparisons, conversion takes place automatically.

If the SIGN clause is not specified for a numeric data description entry whose PICTURE character string contains the character S, then the position of the operational sign is determined as explained in the description of the USAGE clause.

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on one of the proper boundaries in core storage.



The SYNCHRONIZED clause is used to ensure efficiency when performing arithmetic operations on an item.

The SYNCHRONIZED clause may appear only at the elementary level or at level 01. When used at level 01, every elementary item within this scope is synchronized.

If either the LEFT or the RIGHT option is specified, it is treated as comments.

The length of an elementary item is not affected by the SYNCHRONIZED clause.

When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, each occurrence of the item is synchronized.

When the item is aligned, the character positions between the last item assigned and the current item are known as "slack bytes." These unused character positions are included in the size of any group to which the elementary item preceding the synchronized elementary item belongs.

The proper boundary used to align the item to be synchronized depends on the format of the item as defined by the USAGE clause.

When the SYNCHRONIZED clause is specified, the following actions are taken:

For a COMPUTATIONAL item:

1. If its PICTURE is in the range of S9 through S9(4), the item is aligned on a halfword (even) boundary.
2. If its PICTURE is in the range of S9(5) through S9(18), the item is aligned on a fullword (multiple of 4) boundary.

For a COMPUTATIONAL-1 item, the item is aligned on a fullword boundary.

For a COMPUTATIONAL-2 item, the item is aligned on a doubleword (multiple of 8) boundary.

For a DISPLAY or COMPUTATIONAL-3 item, the SYNCHRONIZED clause is treated as comments.

Note: In the discussion that follows, the term "computational" refers to COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items.

When the SYNCHRONIZED clause is specified for an item that also contains a REDEFINES clause, the data item that is redefined must have the proper boundary alignment for the data item that REDEFINES it. For example, if the programmer writes:

```
02 A          PICTURE X(4).
02 B REDEFINES A PICTURE S9(9) COMP SYNC.
```

he must ensure that A begins on a fullword boundary.

When the SYNCHRONIZED clause is specified for a computational item that is the first elementary item subordinate to an item that contains a REDEFINES clause, the computational item must not require the addition of slack bytes.

When SYNCHRONIZED is not specified for binary or internal floating-point items, no space is reserved for slack bytes. However, when computation is done on these fields, the compiler generates the necessary instructions to move the items to a work area which has the correct boundary necessary for computation.

In the File Section, the compiler assumes that all level-01 records containing SYNCHRONIZED items are aligned on a doubleword boundary in the buffer. The user must provide the necessary inter-record slack bytes to ensure alignment.

In the Working-Storage Section, the compiler will align all level-01 entries on a doubleword boundary.

For the purposes of aligning COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items in the Linkage Section, all level-01 items are assumed to begin on doubleword boundaries. Therefore, if the user issues a CALL statement he must ensure that such operands of any USING clause within it are correspondingly aligned.

Slack Bytes

Slack Bytes

There are two types of slack bytes: intra-record slack bytes and inter-record slack bytes.

Intra-record slack bytes are unused character positions preceding each synchronized item in the record.

Inter-record slack bytes are unused character positions added between blocked logical records.

INTRA-RECORD SLACK BYTES: For an output file, or in the Working-Storage Section, the compiler inserts intra-record slack bytes to ensure that all SYNCHRONIZED items are on their proper boundaries. For an input file, or in the Linkage Section, the compiler expects intra-record slack bytes to be present when necessary to assure the proper alignment of a SYNCHRONIZED item.

Because it is important for the user to know the length of the records in a file, the algorithm the compiler uses to determine whether slack bytes are required and, if they are required, the number of slack bytes to add, is as follows:

- The total number of bytes occupied by all elementary data items preceding the computational item are added together, including any slack bytes previously added.

- This sum is divided by m , where:

$m = 2$ for COMPUTATIONAL items of four-digit length or less

$m = 4$ for COMPUTATIONAL items of five-digit length or more

~~$m = 4$ for COMPUTATIONAL-1 items~~

~~$m = 8$ for COMPUTATIONAL-2 items~~

- If the remainder (r) of this division is equal to zero, no slack bytes are required. If the remainder is not equal to zero, the number of slack bytes that must be added is equal to $m - r$.

These slack bytes are added to each record immediately following the elementary data item preceding the computational item. They are defined as if they were an item with a level number equal to that of the elementary item that immediately precedes the SYNCHRONIZED item, and are included in the size of the group which contains them.

For example:

```
01 FIELD-A.
   02 FIELD-B          PICTURE X(5).
   02 FIELD-C.
     03 FIELD-D        PICTURE XX.
     [03 Slack-Bytes   PICTURE X.  Inserted by compiler]
     03 FIELD-E        PICTURE S9(6) COMP SYNC.

01 FIELD-L.
   02 FIELD-M          PICTURE X(5).
   02 FIELD-N          PICTURE XX.
   [02 Slack-Bytes     PICTURE X.  Inserted by compiler]
   02 FIELD-O.
     03 FIELD-P        PICTURE S9(6) COMP SYNC.
```

Slack bytes may also be added by the compiler when a group item is defined with an OCCURS clause and contains within it a SYNCHRONIZED data item with USAGE defined as COMPUTATIONAL. To determine whether slack bytes are to be added, the following action is taken:

- The compiler calculates the size of the group, including all the necessary intra-record slack bytes.
- This sum is divided by the largest m required by any elementary item within the group.
- If r is equal to zero, no slack bytes are required. If r is not equal to zero, $m - r$ slack bytes must be added.

The slack bytes are inserted at the end of each occurrence of the group item containing the OCCURS clause. For example, if a record is defined as follows:

```

01 WORK-RECORD.
   02 WORK-CODE          PICTURE X.
   02 COMP-TABLE OCCURS 10 TIMES.
       03 COMP-TYPE      PICTURE X.
       [03 Ia-Slack-Bytes PICTURE XX.  Inserted by compiler]
       03 COMP-PAY       PICTURE S9(4)V99 COMP SYNC.
       03 COMP-HRS       PICTURE S9(3) COMP SYNC.
       03 COMP-NAME      PICTURE X(5).

```

The record will appear in storage as shown in Figure 17.

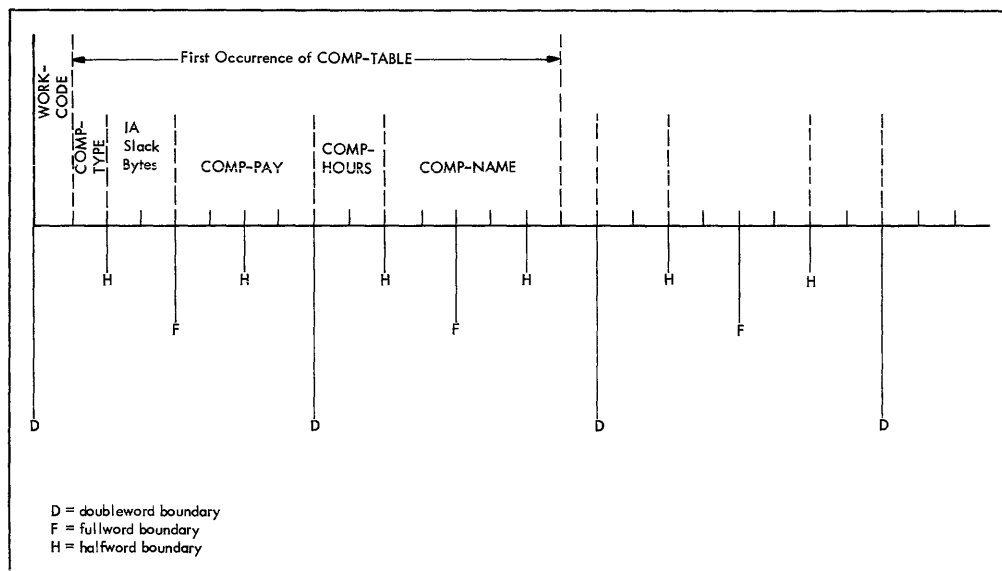


Figure 17. Insertion of the Intra-occurrence Slack Bytes

In order to align COMP-PAY and COMP-HRS upon their proper boundaries, the compiler has added two intra-occurrence slack bytes (shown above as IA-SLACK-BYTES).

However, without further adjustment, the second occurrence of COMP-TABLE would now begin one byte before a doubleword boundary, and the alignment of COMP-PAY and COMP-HRS would not be valid for any occurrence of the table after the first. Therefore, the compiler must add inter-occurrence slack bytes at the end of the group, as though the record had been written:

Slack Bytes

```

01 WORK-RECORD.
   02 WORK-CODE          PICTURE X.
   02 COMP-TABLE OCCURS 10 TIMES.
     03 COMP-TYPE        PICTURE X.
     [03 Ia-Slack-Bytes  PICTURE XX.  Inserted by compiler]
     03 COMP-PAY         PICTURE S9(4)V99 COMP SYNC.
     03 COMP-HRS        PICTURE S9(3) COMP SYNC.
     03 COMP-NAME       PICTURE X(5).
     [03 Ie-Slack-Bytes  PICTURE XX.  Inserted by compiler]

```

so that the second (and each succeeding) occurrence of COMP-TABLE begins one byte beyond a doubleword boundary. The storage layout for the first occurrences of COMP-TABLE will now appear as shown in Figure 18.

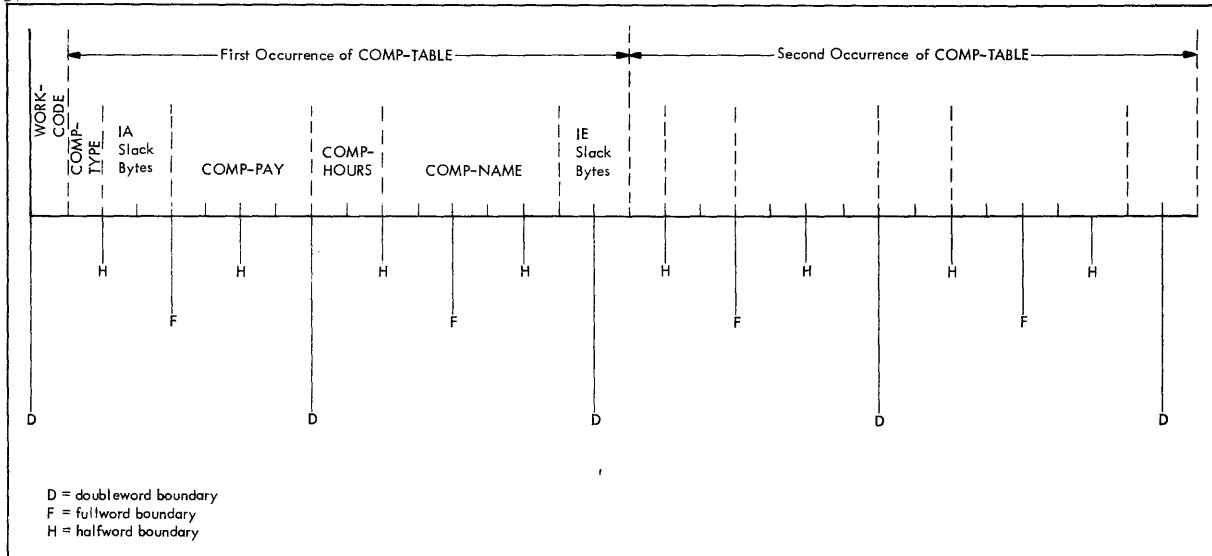


Figure 18. Insertion of Inter-occurrence Slack Bytes

Each succeeding occurrence within the table will now begin at the same relative position to word boundaries as the first.

Where SYNCHRONIZED data items defined as COMPUTATIONAL, COMPUTATIONAL-1 or COMPUTATIONAL-2 follow an entry containing an OCCURS clause with the DEPENDING ON option, slack bytes are added on the basis of the field occurring the maximum number of times. If the length of this field is not divisible by the m required for the computational data, only certain values of the data-name that is the object of the DEPENDING ON option will give proper alignment of the computational fields. These values are those for which the length of the field times the number of occurrences plus the slack bytes that have been calculated based on the maximum number of occurrences is divisible by m.

For example:

```

01 FIELD-A.
   02 FIELD-B          PICTURE 99.
   02 FIELD-C          PICTURE X OCCURS 20 TO 99 TIMES
     DEPENDING ON FIELD-B.
     [02 Slack-Byte    PICTURE X.  Inserted by compiler]
     02 FIELD-D        PICTURE S99 COMP SYNC.

```

In this example, when references to FIELD-D are required, FIELD-B is restricted to odd values only.

```

01 FIELD-A.
   02 FIELD-B          PICTURE 999.
   02 FIELD-C          PICTURE XX OCCURS 20 TO 99 TIMES
                        DEPENDING ON FIELD-B.
[02 Slack-Byte       PICTURE X. Inserted by compiler]
   02 FIELD-D          PICTURE S99 COMP SYNC.

```

In this example all values of FIELD-B give proper references to FIELD-D.

INTER-RECORD SLACK BYTES: If the file contains blocked logical records that are to be processed in a buffer, and any of the records contain entries defined as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2, for which the SYNCHRONIZED clause is specified, the user must add any inter-record slack bytes needed for proper alignment.

The lengths of all the elementary data items in the record, including all intra-record slack bytes, are added. For mode V records, it is necessary to add four bytes for the count field. The total is then divided by the highest value of m for any one of the elementary items in the record.

If r (the remainder) is equal to zero, no inter-record slack bytes are required. If r is not equal to zero, $m - r$ slack bytes are required. These slack bytes may be specified by writing a level-02 FILLER at the end of the record.

Example: The following example shows the method of calculating both intra-record and inter-record slack bytes. Consider the following record description:

```

01 COMP-RECORD.
   02 A-1      PICTURE X(5).
   02 A-2      PICTURE X(3).
   02 A-3      PICTURE X(3).
   02 B-1      PICTURE S9999  USAGE COMP SYNCHRONIZED.
   02 B-2      PICTURE S99999  USAGE COMP SYNCHRONIZED.
   02 B-3      PICTURE S9999  USAGE COMP SYNCHRONIZED.

```

The number of bytes in A-1, A-2, and A-3 total 11. B-1 is a 4-digit COMPUTATIONAL item and, therefore, one intra-record slack byte must be added before B-1. With this byte added, the number of bytes preceding B-2 total 14. Since B-2 is a COMPUTATIONAL item of 5 digits in length, two intra-record slack bytes must be added before it. No slack bytes are needed before B-3.

The revised record description entry now appears as:

```

01 COMP-RECORD.
   02 A-1      PICTURE X(5).
   02 A-2      PICTURE X(3).
   02 A-3      PICTURE X(3).
[02 Slack-Byte-1 PICTURE X. Inserted by compiler]
   02 B-1      PICTURE S9999  USAGE COMP SYNCHRONIZED.
[02 Slack-Byte-2 PICTURE XX. Inserted by compiler]
   02 B-2      PICTURE S99999  USAGE COMP SYNCHRONIZED.
   02 B-3      PICTURE S9999  USAGE COMP SYNCHRONIZED.

```

There are a total of 22 bytes in COMP-RECORD, but from the rules given in the preceding discussion, it appears that $m = 4$ and $r = 2$.

USAGE Clause

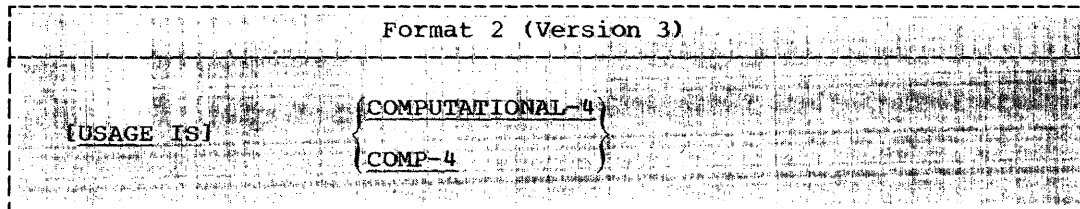
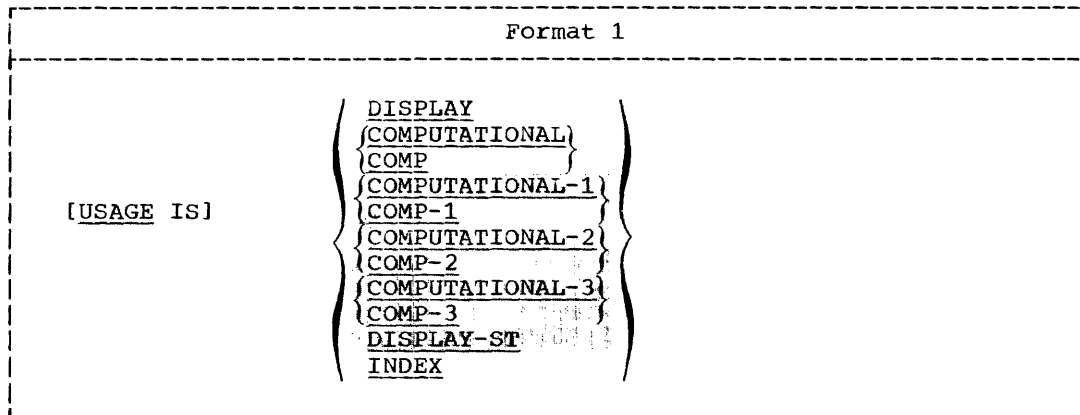
Therefore, to attain proper alignment for blocked records, the user must add two inter-record slack bytes at the end of the record.

The final record description entry appears as:

```
01 COMP-RECORD.
  02 A-1          PICTURE X(5).
  02 A-2          PICTURE X(3).
  02 A-3          PICTURE X(3).
[02 Slack-Byte-1 PICTURE X.  Inserted by compiler]
  02 B-1          PICTURE S9999  USAGE COMP SYNCHRONIZED.
[02 Slack-Byte-2 PICTURE XX.  Inserted by compiler]
  02 B-2          PICTURE S99999  USAGE COMP SYNCHRONIZED.
  02 B-3          PICTURE S9999  USAGE COMP SYNCHRONIZED.
  02 FILLER       PICTURE XX. [inter-record slack bytes added by
                             user]
```

USAGE Clause

The USAGE clause specifies the manner in which a data item is represented in core storage.



The USAGE clause can be specified at any level of data description. However, if the USAGE clause is written at a group level, it applies to each elementary item in the group. The usage of an elementary item cannot contradict the usage of a group to which an elementary item belongs.

This clause specifies the manner in which a data item is represented in core storage. However, the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operand referred to.

If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, it is assumed that the usage is DISPLAY.

Note: ASCII considerations for the USAGE clause are given in Appendix E.

DISPLAY OPTION

The DISPLAY option can be explicit or implicit. It specifies that the data item is stored in character form, one character per eight-bit byte. This corresponds to the form in which information is represented for initial card input or for final printed or punched output. USAGE IS DISPLAY is valid for the following types of items:

- alphabetic
- alphanumeric
- alphanumeric edited
- numeric edited
- external decimal
- external floating point

The alphabetic, alphanumeric, alphanumeric edited, and numeric edited items are discussed in the description of the PICTURE clause.

External Decimal Items: These items are sometimes referred to as zoned decimal items. Each digit of a number is represented by a single byte. The four high-order bits of each byte are zone bits; the four high-order bits of the low-order byte represent the sign of the item. The four low-order bits of each byte contain the value of the digit. When external decimal items are used for computations, the compiler performs the necessary conversions.

The maximum length of an external decimal item is 18 digits.

Examples of external decimal items and their internal representation are shown in Figure 19.

External Floating-Point Items: The PICTURE of an external floating-point item is in the following form:

{+}mantissa E{+}exponent

(See the discussion of the PICTURE clause for the valid combination of symbols.)

The mantissa is the decimal part of the number.

The exponent specifies a power of ten that is used as a multiplier.

The value of an external floating-point number is the mantissa multiplied by the power of ten expressed by the exponent. The magnitude of a number represented by a floating-point item must be greater than 5.4×10^{-79} but must not exceed $.72 \times 10^{76}$.

When used as a numeric operand an external floating-point number is scanned at object time, and converted to the equivalent internal floating-point values. In this form, the number is used in arithmetic operations. (See COMPUTATIONAL-1 and COMPUTATIONAL-2 options.)

USAGE Clause

An example of an external floating-point number and its internal representation is shown in Figure 19.

The Computational Options

A COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3, or COMPUTATIONAL-4 item represents a value to be used in arithmetic operations and must be numeric. If the USAGE of any group item is described with any of these options, it is the elementary items within this group which have that USAGE. The group item itself cannot be used in computations.

COMPUTATIONAL OPTION: This option is specified for binary data items. Such items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign.

The amount of storage occupied by a binary item depends on the number of decimal digits defined in its PICTURE clause:

<u>Digits in PICTURE Clause</u>	<u>Storage Occupied</u>
1 through 4	2 bytes (halfword)
5 through 9	4 bytes (fullword)
10 through 18	8 bytes (2 fullwords -- not necessarily a doubleword)

The leftmost bit of the storage area is the operational sign.

The PICTURE of a COMPUTATIONAL item may contain only 9's, the operational sign character S, the implied decimal point V, and one or more P's.

An example of a binary item is shown in Figure 19.

Note: The COMPUTATIONAL option is system dependent and normally is assigned to representations that yield the greatest efficiency when performing arithmetic operations on that system; for this compiler, the COMPUTATIONAL option is binary.

COMPUTATIONAL-1, COMPUTATIONAL-2 OPTIONS: These options are specified for internal floating-point items. Such an item is equivalent to an external floating-point item in capability and purpose. Such items occupy either 4 or 8 bytes of storage.

The sign of the fraction (mantissa) is the leftmost bit in either format.

The exponent appears in bit positions 1 through 7.

The fraction -- equivalent to the mantissa -- appears in the rightmost bytes.

COMPUTATIONAL-1 is specified for short-precision internal floating-point items. Such items are four bytes in length, aligned on a fullword boundary. The fraction occupies the rightmost three bytes of the item.

COMPUTATIONAL-2 is specified for long-precision floating-point items. Such items are eight bytes in length, and are aligned on a doubleword boundary. The fraction occupies the rightmost seven bytes of the item.

If a VALUE clause is associated with an internal floating-point item, the literal must be a floating-point literal (for example, VALUE IS 7.14E92).

No PICTURE clause may be associated with an internal floating-point item.

Examples of internal floating-point items, and their internal representation, are shown in Figure 19.

COMPUTATIONAL-3 OPTION: This option is specified for internal decimal items. Such an item appears in storage in packed decimal format. There are two digits per byte, with the sign contained in the low-order four bits of the rightmost byte. Such an item may contain any of the digits 0 through 9, plus a sign, representing a value not exceeding 18 decimal digits.

For internal decimal items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration that is interpreted as positive, but that does not represent an overpunch.

The PICTURE of a COMPUTATIONAL-3 item may contain only 9's, the operational sign character S, the assumed decimal point V, and one or more P's.

Examples of internal-decimal items and their internal representation are shown in Figure 19.

Program Product Information -- Version 3

COMPUTATIONAL-4 OPTION: This option (Format 2) is specified for system-independent binary items. For this compiler, it is the equivalent of COMPUTATIONAL.

USAGE DISPLAY-ST is discussed in the chapter on Sterling Currency. USAGE INDEX is discussed in the chapter on Table Handling.

Item	Value	Description	Internal Representation*
External Decimal	-1234	DISPLAY PICTURE 9999	<pre> Z1 Z2 Z3 F4 ----- byte </pre>
		DISPLAY PICTURE S9999	<pre> Z1 Z2 Z3 D4 ----- byte </pre> <p>Note that, internally, the D4, which represents -4, is the same bit configuration as the EBCDIC character M.</p>
		(Version 3) DISPLAY PICTURE S9999 SIGN LEADING	<pre> D1 Z2 Z3 Z4 ----- byte </pre> <p>Note that internally the D1, which represents -1, is the same bit configuration as the EBCDIC character J.</p>
		(Version 3) DISPLAY PICTURE S9999 SIGN TRAILING SEPARATE	<pre> Z1 Z2 Z3 Z4 - ----- byte </pre>
Binary	-1234	COMPUTATIONAL PICTURE S9999	<pre> 1111 1011 0010 1110 ----- ↑ byte S </pre>
		(Version 3) COMP-4 PICTURE S9999	<p>Note that, internally, negative binary numbers appear in two's complement form.</p>

Figure 19. Internal Representation of Numeric Items (Part 1 of 2)

Item	Value	Description	Internal Representation*																				
Internal Decimal	+1234	COMPUTATIONAL-3 PICTURE 9999	<table border="1"> <tr> <td>01</td> <td>23</td> <td>4P</td> </tr> <tr> <td colspan="3">byte</td> </tr> </table>	01	23	4P	byte																
		01	23	4P																			
byte																							
COMPUTATIONAL-3 PICTURE S9999	<table border="1"> <tr> <td>01</td> <td>23</td> <td>4C</td> </tr> <tr> <td colspan="3">byte</td> </tr> </table>	01	23	4C	byte																		
01	23	4C																					
byte																							
External Floating-Point	+12.34E+2	DISPLAY PICTURE +99.99E-99	<table border="1"> <tr> <td>+</td> <td>1</td> <td>2</td> <td>.</td> <td>3</td> <td>4</td> <td>E</td> <td>D</td> <td>0</td> <td>2</td> </tr> <tr> <td colspan="10">byte</td> </tr> </table>	+	1	2	.	3	4	E	D	0	2	byte									
+	1	2	.	3	4	E	D	0	2														
byte																							
Internal Floating-Point		COMPUTATIONAL-1	<table border="1"> <tr> <td>S</td> <td>Exponent</td> <td>Fraction</td> </tr> <tr> <td>0 1</td> <td>7 8</td> <td>31</td> </tr> </table>	S	Exponent	Fraction	0 1	7 8	31														
		S	Exponent	Fraction																			
0 1	7 8	31																					
COMPUTATIONAL-2	<table border="1"> <tr> <td>S</td> <td>Exponent</td> <td>Fraction</td> </tr> <tr> <td>0 1</td> <td>7 8</td> <td>63</td> </tr> </table>	S	Exponent	Fraction	0 1	7 8	63																
S	Exponent	Fraction																					
0 1	7 8	63																					
<p>*Codes used in this column are as follows: Z = zone, equivalent to hexadecimal F, bit configuration 1111</p> <p>Hexadecimal numbers and their equivalent meanings are: F = nonprinting plus sign (treated as an absolute value) C = internal equivalent of plus sign, bit configuration 1100 D = internal equivalent of minus sign, bit configuration 1101</p> <p>S = sign position of a numeric field; internally, 1 in this position means the number is negative 0 in this position means the number is positive</p> <p>b = a blank</p>																							

Figure 19. Internal Representation of Numeric Items (Part 2 of 2)

VALUE Clause

VALUE Clause

The VALUE clause is used to define the initial value of a Working-Storage item or the value associated with a condition-name.

There are two formats of the VALUE clause:

```
-----  
Format 1  
-----  
VALUE IS literal  
-----
```

```
-----  
Format 2  
-----  
{ VALUE IS }  
{ VALUES ARE } literal-1 [THRU literal-2]  
[literal-3 [THRU literal-4]]...
```

The VALUE clause must not be stated for any item whose size, explicit or implicit, is variable.

A figurative constant may be substituted wherever a literal is specified.

Rules governing the use of the VALUE clause differ with the particular section of the Data Division in which it is specified.

1. In the File Section and the Linkage Section, the VALUE clause must be used only in condition-name entries. However, this compiler will accept the VALUE clause in other File Section and Linkage Section entries, and treat it as comments.
2. In the Working-Storage Section, the VALUE clause must be used in condition-name entries, and it may also be used to specify the initial value of any data item. It causes the item to assume the specified value at the start of execution of the object program. If the VALUE clause is not used in an item's description, the initial value is unpredictable.
3. In the Report Section, the VALUE clause causes the report data item to assume the specified value each time its report group is presented. This clause may be used only at an elementary level in the Report Section. The Report Section is discussed in detail in the "Report Writer" chapter.

The VALUE clause must not be specified in a data description entry that contains an OCCURS clause or in an entry that is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries.

Within a given record description, the VALUE clause must not be used in a data description entry that is subsequent to a data description entry which contains an OCCURS clause with a DEPENDING ON phrase.

VALUE Clause

The VALUE clause must not be specified in a data description entry which contains a REDEFINES clause or in an entry which is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal. The VALUE clause then cannot be specified at subordinate levels within this group.

The VALUE clause cannot be specified for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

The VALUE clause must not be specified for external floating-point items.

The following rules apply:

1. If the item is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves, except that the literal must not have a value that would require truncation of nonzero digits.

If the item is an internal floating-point item, the literal must be a floating-point literal (for example VALUE IS 7.14E+2).

2. If the item is alphabetic or alphanumeric (elementary or group), all literals in the VALUE clause must be nonnumeric literals. The literal is aligned according to the alignment rules (see "JUSTIFIED Clause"), except that the number of characters in the literal must not exceed the size of the item.
3. All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be within the range 1000 through 99000 or zero. For PICTURE PPP99, the literal must be within the range .00000 through .00099.
4. The function of the editing characters in a PICTURE clause is ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item.

Format 1 of the VALUE clause must not conflict with other clauses either in the data description of the item or in the data descriptions within the hierarchy of this term.

Format 2 of the VALUE clause is used to describe a condition-name. Each condition-name requires a separate level-88 entry. A Format 2 VALUE clause associates a value, values, or range of values with the condition-name. In a condition-name entry, the VALUE clause is required and is the only clause permitted in the entry.

A condition-name is a name assigned by the user to the values a data item may assume during object program execution. A condition-name must be formed according to the rules for data-name formation. The condition-name entries for a particular conditional variable must follow the conditional variable. Hence, a level-88 entry must always be preceded either by the entry for the conditional variable or by another level-88 entry (in the case of several consecutive condition-names pertaining to a given item).

RENAMES Clause

The THRU option assigns a range of values to a condition-name. Wherever used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

The type of literal in a condition-name entry must be consistent with the data type of the conditional variable. In the following example, CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables; the associated condition-names immediately follow the level-number 88. The PICTURE associated with COUNTY-NO limits the condition-name value to a 2-digit numeric literal. The PICTURE associated with CITY limits the condition-name value to a 3-character nonnumeric literal. Any values for the condition-names associated with CITY-COUNTY-INFO cannot exceed 5 characters, and the literal (since this is a group item) must be nonnumeric:

```
05 CITY-COUNTY-INFO.,
    88 BRONX                VALUE "03NYC".
    88 BROOKLYN            VALUE "24NYC".
    88 MANHATTAN           VALUE "31NYC".
    88 QUEENS              VALUE "41NYC".
    88 STATEN-ISLAND      VALUE "43NYC".
10 COUNTY-NO              PICTURE 99.
    88 DUTCHESS            VALUE 14.
    88 KINGS               VALUE 24.
    88 NEW-YORK            VALUE 31.
    88 RICHMOND           VALUE 43.
10 CITY                  PICTURE X(3).
    88 BUFFALO             VALUE "BUF".
    88 NEW-YORK-CITY      VALUE "NYC".
    88 POUGHKEEPSIE       VALUE "POK".
05 POPULATION ...
```

Every condition-name pertains to an item in such a way that the condition-name may be qualified by the name of the item and the item's qualifiers. The use of condition-names in conditions is described in "Conditions."

A condition-name may pertain to an item (a conditional variable) requiring subscripts. In this case, the condition-name, when written in the Procedure Division, must be subscripted according to the requirements of the associated conditional variable.

A condition-name can be associated with any elementary or group item except the following:

1. A level-66 item
2. A group containing items with descriptions which include JUSTIFIED, SYNCHRONIZED, or USAGE other than DISPLAY
3. An index data item (see "Table Handling")

RENAMES Clause

The RENAMES clause permits alternate, possibly overlapping, groupings of elementary data.

```
-----
                        Format
-----
66 data-name-1 RENAMES data-name-2 [THRU data-name-3]
-----
```


One or more RENAMES entries can be written for a logical record.

All RENAMES entries associated with a given logical record must immediately follow its last data description entry.

Data-name-2 and data-name-3 must be the names of elementary items or groups of elementary items in the associated logical record and cannot be the same data name. data-name-3 cannot be subordinate to data-name-2.

When data-name-3 is not specified, data-name-2 can be either a group item or an elementary item. When data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

When data-name-3 is specified, data-name-1 is a group item that includes all elementary items:

1. starting with data-name-2 (if it is an elementary item); or starting with the first elementary item within data-name-2 (if it is a group item), and
2. ending with data-name-3 (if it is an elementary item); or ending with the last elementary item within data-name-3 (if it is a group item).

A level-66 entry cannot rename another level-66 entry nor can it rename a level-77, level-88, or level-01 entry.

data-name-1 cannot be used as a qualifier and can be qualified only by the names of the level-01 or FD entries.

Both data-name-2 and data-name-3 may be qualified.

Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor may either of them be subordinate to an item that has an OCCURS clause in its data description entry.

data-name-2 must precede data-name-3 in the record description; after any associated redefinition, the beginning point of the area described by data-name-3 must logically follow the beginning point of the area described by data-name-2.

For example, the following Working-Storage record is incorrect:

```

01 ERR-REC.
  02 GROUP-A.
    03 FIELD-1A.
      04 ITEM-1A PICTURE XXXX.
      04 ITEM-2A PICTURE XXX.
    03 FIELD-2A.
      04 ITEM-3A PICTURE XXX.
      04 ITEM-4A PICTURE XXX.
  02 GROUP-B REDEFINES GROUP-A.
    03 FIELD-1B.
      04 ITEM-1B PICTURE XX.
      04 ITEM-2B PICTURE XXX.
      04 ITEM-3B PICTURE XX.
    03 FIELD-2B.
      04 ITEM-4B PICTURE XX.
      04 ITEM-5B PICTURE XX.
      04 ITEM-6B PICTURE XX.
66 NEW-ERR-REC RENAMES ITEM-3A THRU ITEM-2B.

```

Although ITEM-3A precedes ITEM-2B in the record description, ITEM-2B logically precedes ITEM-3A in storage. Thus, this example is incorrect.

RENAMES Clause

The following shows the corrected Working-Storage record:

```
01 CORRECTED-RECORD.
02 GROUP-A.
    03 FIELD-1A.
        04 ITEM-1A PICTURE XX.
        04 ITEM-2A PICTURE XXX.
        04 ITEM-3A PICTURE XX.
    03 FIELD-2A.
        04 ITEM-4A PICTURE XX.
        04 ITEM-5A PICTURE XX.
        04 ITEM-6A PICTURE XX.
02 GROUP-B REDEFINES GROUP-A.
    03 FIELD-1B.
        04 ITEM-1B PICTURE XXXX.
        04 ITEM-2B PICTURE XXX.
    03 FIELD-2B.
        04 ITEM-3B PICTURE XXX.
        04 ITEM-4B PICTURE XXX.
66 NEW-REC RENAMES ITEM-2A THRU ITEM-3B.
```

In this example ITEM-2A precedes ITEM-3B both in the record description and logically in storage.

The following example shows how the RENAMES clause might be used in an actual program:

```
01 OUT-REC.
02 FIELD-X.
    03 SUMMARY-GROUPX.
        04 FILE-1 PICTURE X.
        04 FILE-2 PICTURE X.
        04 FILE-3 PICTURE X.
02 FIELD-Y.
    03 SUMMARY-GROUPY.
        04 FILE-1 PICTURE X
        04 FILE-2 PICTURE X.
        04 FILE-3 PICTURE X.
02 FIELD-Z.
    03 SUMMARY-GROUPZ.
        04 FILE-1 PICTURE X.
        04 FILE-2 PICTURE X.
        04 FILE-3 PICTURE X.
66 SUM-X RENAMES FIELD-X.
66 SUM-XY RENAMES FIELD-X THRU FIELD-Y.
66 SUM-XYZ RENAMES FIELD-X THRU FIELD-Z.
```

In the Procedure Division, the programmer may wish, for example, to do a complete tally of files in each field of the foregoing record. If all active files are represented by an A and all inactive files are represented by an I, the statement

```
EXAMINE SUM-XYZ TALLYING ALL "A"
```

would accomplish this purpose. The two additional RENAMES entries (SUM-X and SUM-XY) allow a less inclusive tally, if desired. (The EXAMINE statement is discussed in "Procedure Division.")

PART IV -- PROCEDURE DIVISION

- ORGANIZATION OF THE PROCEDURE DIVISION

- ARITHMETIC EXPRESSIONS

- CONDITIONS

- CONDITIONAL STATEMENTS

- DECLARATIVES

- ARITHMETIC STATEMENTS

- PROCEDURE BRANCHING STATEMENTS

- DATA-MANIPULATION STATEMENTS

- INPUT/OUTPUT STATEMENTS

- SUBPROGRAM LINKAGE STATEMENTS

- COMPILER-DIRECTING STATEMENTS

ORGANIZATION OF THE PROCEDURE DIVISION

The Procedure Division contains the specific instructions for solving a data processing problem. COBOL instructions are written in statements, which may be combined to form sentences. Groups of sentences may form paragraphs, and paragraphs may be combined to form sections.

The Procedure Division must begin with the header PROCEDURE DIVISION followed by a period and a space unless Subprogram Linkage is used. In this case, the Procedure Division header in a called program may optionally contain a USING clause preceding the period (see "Subprogram Linkage").

The Procedure Division header is followed, optionally, by Declarative Sections, which are in turn followed by procedures, each made up of statements, sentences, paragraphs, and/or sections, in a syntactically valid format. The end of the Procedure Division (and the physical end of the program) is that physical position in a COBOL source program after which no further procedures appear.

The statement is the basic unit of the Procedure Division. A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb. There are three types of statements: conditional statements containing conditional expressions (that is, tests for a given condition), imperative statements consisting of an imperative verb and its operands, and compiler-directing statements consisting of a compiler-directing verb and its operands.

A sentence is composed of one or more statements. The statements may optionally be separated by semicolons or the word AND. A sentence must be terminated by a period followed by a space.

Several sentences that convey one idea or procedure may be grouped to form a paragraph. A paragraph must begin with a paragraph-name followed by a period. A paragraph may be composed of one or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name, at the end of the Procedure Division, or, in the Declarative portion, at the key words END DECLARATIVES.

One or more paragraphs form a section. A section must begin with a section header (section-name followed by the word SECTION, followed by a period; if program segmentation is desired, a space and a priority number followed by a period may be inserted after the word SECTION). The general term procedure-name may refer to both paragraph-names and section-names.

The Procedure Division may contain both declaratives and procedures.

Declarative sections must be grouped at the beginning of the Procedure Division, preceded by the key word DECLARATIVES followed by a period and a space. Declarative sections are concluded by the key words END DECLARATIVES followed by a period and a space. (For a more complete discussion of declarative sections, see "Declaratives.")

A procedure is composed of a paragraph or group of successive paragraphs, or a section or group of successive sections within the Procedure Division. Paragraphs need not be grouped into sections.

If sections are used within the Procedure Division, a section header should immediately follow the Procedure Division header, except when a declarative section is included, in which case the section header should immediately follow END DECLARATIVES.

A section ends immediately before the next section-name or at the end of the Procedure Division, or, in the Declarative portion of the Procedure Division, immediately before the next section-name or at the words END DECLARATIVES, where END must appear in Area A.

If program segmentation is used, the programmer must divide the entire Procedure Division into named sections. Program segmentation is discussed in "Segmentation."

Execution begins with the first statement of the Procedure Division, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules in this chapter indicate some other order.

Structure of the Procedure Division

```
PROCEDURE DIVISION [REDACTED].  
  [(DECLARATIVES.  
    {section-name SECTION. USE Sentence.  
    {paragraph-name. {sentence}...}...}  
  END DECLARATIVES.]  
  {section-name SECTION [priority].]  
  {paragraph-name. {sentence}...}...}
```

CATEGORIES OF STATEMENTS

There are three categories of statements used in COBOL: conditional statements, imperative statements, and compiler-directing statements.

A conditional statement is a statement containing a condition that is tested (see "Conditions") to determine which of the alternate paths of program flow is to be taken.

An imperative statement specifies that an unconditional action is to be taken by an object program. An imperative statement may also consist of a series of imperative statements.

A compiler-directing statement directs the compiler to take certain actions at compile time.

CONDITIONAL STATEMENTS

COBOL statements used as conditional statements are:

IF		
ON		
ADD	}	(ON SIZE ERROR)
COMPUTE		
SUBTRACT		
MULTIPLY		
DIVIDE		
GO TO		(DEPENDING ON)
READ	}	(AT END)
SEARCH		
RETURN		
WRITE		(AT END-OF-PAGE)
READ	}	(INVALID KEY)
SEARCH		
WRITE		
PERFORM		
PERFORM		(UNTIL)
SEARCH		(WHEN)
EXIT		

The options in parentheses cause otherwise imperative statements to be treated as conditionals at execution time. A discussion of these options is included as part of the description of the associated imperative statement.

IMPERATIVE STATEMENTS

COBOL verbs used in imperative statements can be grouped into the following categories and subcategories:

A. DECLARATIVES

USE

B. PROCEDURAL1. Arithmetic

ADD
COMPUTE
DIVIDE
MULTIPLY
SUBTRACT

2. Procedure Branching

GO TO
ALTER
PERFORM
STOP
EXIT

3. Data-Manipulation

MOVE
EXAMINE

Statement Categories

4. Input/Output

OPEN
CLOSE
SEEK
READ
WRITE
ACCEPT
DISPLAY
CLOSE

5. Report Writer

GENERATE
INITIATE
TERMINATE

6. Table Handling

SEARCH
SET

7. Sort

SORT
RETURN
RELEASE

Debugging
READY TRACE
RESUME TRACE
STOP

Note: Report Writer, Table Handling, and Sort statements are discussed in separate chapters.

CALL
ENTER
NOTE

COMPILER-DIRECTING STATEMENTS

COBOL verbs used in compiler-directing statements are:

COPY
ENTER
NOTE

Note: The COPY statement is discussed in "Source Program Library Facility." The statements listed in "Format Control of the Source Program Listing" and "Extended Source Program Library Facility" are also considered as compiler-directing statements.

ARITHMETIC EXPRESSIONS

Arithmetic expressions are used as operands of certain conditional and arithmetic statements.

An arithmetic expression may consist of any of the following:

1. an identifier described as a numeric elementary item
2. a numeric literal
3. identifiers and literals, as defined in items 1 and 2, separated by arithmetic operators
4. two arithmetic expressions, as defined in items 1, 2, and/or 3, separated by an arithmetic operator
5. an arithmetic expression, as defined in items 1, 2, 3, and/or 4, enclosed in parentheses

Any arithmetic expression may be preceded by a unary `-`.

ARITHMETIC OPERATORS

There are five arithmetic operators that may be used in arithmetic expressions. Each is represented by a specific character or character combination that must be preceded by a space and followed by a space, except that a unary operator must not be preceded by a space when it follows a left parenthesis. This completes the list of operators and the unary operator to be omitted.

<u>Arithmetic Operator</u>	<u>Meaning</u>
<code>+</code>	addition
<code>-</code>	subtraction
<code>*</code>	multiplication
<code>/</code>	division
<code>**</code>	exponentiation

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated.

Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the least inclusive to the most inclusive set.

When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order is implied:

1. unary `-`
2. `**`
3. `*` and `/`
4. `+` and `-`

When the order of consecutive operations on the same hierarchical level is not completely specified by parentheses, the order of operation is from left to right.

Arithmetic Symbol Pairs

Figure 20 shows permissible symbol pairs. A symbol pair in an arithmetic expression is the occurrence of two symbols that appear in sequence.

First Symbol \ Second Symbol	Variable (identifier or literal)	* / ** + -	unary - unary -	()
Variable (identifier or literal)	-	p	p	-	p
* / ** + -	p	-	p	p	-
unary - unary -	p	-	-	p	-
(p	-	p	p	-
)	-	p	p	-	p

p indicates a permissible pairing
 - indicates that the pairing is not permitted

Figure 20. Permissible Symbol Pairs -- Arithmetic Expressions

An arithmetic expression may begin only with a left parenthesis, a ~~unary -~~ unary -, or a variable, and may end only with a right parenthesis or a variable.

There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression.

CONDITIONS

A condition causes the object program to select between alternate paths of control depending on the truth value of a test. Conditions are used in IF, PERFORM, and SEARCH statements.

A condition is one of the following:

- class condition
- condition-name condition
- relation condition
- sign condition
- switch-status condition

In addition, there are two constructions that affect the evaluation of conditions. These are:

1. (condition)

 Parentheses may be used to group conditions (see "Compound Conditions").

2. NOT condition

 The construction -- NOT condition -- (where condition is one of the five conditions listed above) is not permitted if the condition itself contains a NOT.

Conditions may be combined through the use of logical operators to form compound conditions (for a full discussion, see "Compound Conditions").

TEST CONDITIONS

A test condition is an expression that, taken as a whole, may be either true or false, depending on the circumstances existing when the expression is evaluated.

There are five types of simple conditions which, when preceded by the word IF, constitute one of the five types of tests: class test, condition-name test, relation test, sign test, and switch-status test.

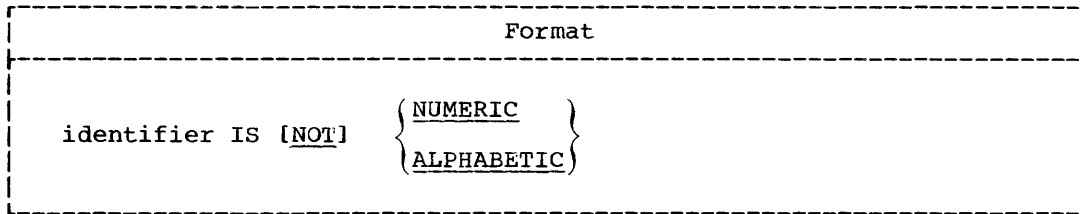
The construction -- NOT condition -- may be used in any simple test condition to make the relation specify the opposite of what it would express without the word NOT. For example, NOT (AGE GREATER THAN 21) is the opposite of AGE GREATER THAN 21.

Each of the previously mentioned tests, when used within the IF statement, constitutes a conditional statement (see "Conditional Statements").

Class Condition

Class Condition

The class test determines whether data is alphabetic or numeric.



The operand being tested must be described implicitly or explicitly as USAGE DISPLAY or USAGE COMPUTATIONAL 3.

A numeric data item consists of the digits 0 through 9, with or without an operational sign.

The identifier being tested is determined to be numeric only if the contents consist of any combination of the digits 0 through 9. If the PICTURE of the identifier being tested does not contain an operational sign, the identifier being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If its PICTURE does contain an operational sign, the identifier being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present. Valid operational signs are hexadecimal F, C, and D.

Program Product Information -- Version 3



The NUMERIC test cannot be used with an identifier described as alphabetic.

An alphabetic data item consists of the space character and the characters A through Z.

The identifier being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

The ALPHABETIC test cannot be used with an identifier described as numeric.

Figure 21 shows valid forms of the class test.

Type of Identifier	Valid Forms of the Class Test	
Alphabetic	ALPHABETIC	NOT ALPHABETIC
Alphanumeric, Alphanumeric Edited, or Numeric Edited	ALPHABETIC NUMERIC	NOT ALPHABETIC NOT NUMERIC
External-Decimal or Internal-Decimal	NUMERIC	NOT NUMERIC

Figure 21. Valid Forms of the Class Test

Condition-Name Condition

The condition-name condition causes a conditional variable to be tested to determine whether or not its value is equal to one of the values associated with condition-name.

Format
condition-name

An example of the use of the condition-name condition follows:

```
02 MARITAL-STATUS PICTURE 9.
   88 SINGLE VALUE 1.
   88 MARRIED VALUE 2.
   88 DIVORCED VALUE 3.
```

MARITAL-STATUS is the conditional variable; SINGLE, MARRIED, and DIVORCED are condition-names. Only one of the conditions specified by condition-name can be present for individual records in the file. To determine the marital status of the individual whose record is being processed, IF SINGLE ... can be coded, and its evaluation as true or false determines the subsequent path the object program takes.

A condition-name is used in conditions as an abbreviation for the relation condition, since the associated condition-name is equal to only one of the values (or ranges of values) assigned to that conditional variable. That is, to determine whether the condition SINGLE is present, IF MARITAL-STATUS = 1 ... would have the same effect as using the condition-name test IF SINGLE

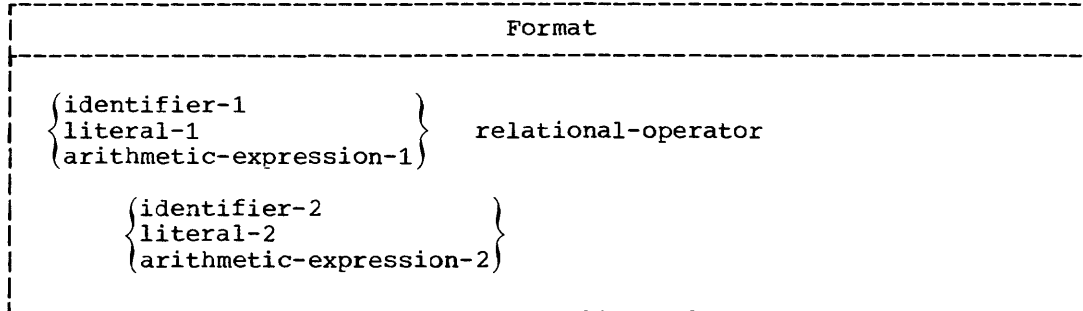
If the condition-name is associated with a range of values (or with several ranges of values), the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

(An example of both group and elementary condition-name entries is given in the description of the VALUE clause in "Data Division".)

Relation Condition

Relation Condition

A relation condition causes a comparison of two operands, either of which may be an identifier, a literal, or an arithmetic expression.



The first operand is called the subject of the condition; the second operand is called the object of the condition.

The subject and object may not both be literals.

The subject and object must have the same USAGE, except when two numeric operands are compared.

A relational-operator specifies the type of comparison to be made in a relation condition. The meaning of the relational operators is shown in Figure 22.

Relational-operator	Meaning
IS [NOT] <u>GREATER THAN</u> IS [NOT] >	Greater than or not greater than
IS [NOT] <u>LESS THAN</u> IS [NOT] <	Less than or not less than
IS [NOT] <u>EQUAL TO</u> IS [NOT] =	Equal to or not equal to

Figure 22. Relational-operators and Their Meanings

The word TO in the EQUAL TO relational operator is required, however, ~~the word TO may be omitted.~~

The relational-operator must be preceded by, and followed by, a space.

COMPARISON OF NUMERIC OPERANDS: For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands.

Zero is considered a unique value, regardless of sign.

Comparison of numeric operands is permitted regardless of the manner in which their USAGE is described.

Unsigned numeric operands are considered positive for purposes of comparison.

COMPARISON OF NONNUMERIC OPERANDS: For nonnumeric operands, or for one numeric and one nonnumeric operand, a comparison is made with respect to the binary collating sequence of the characters in the EBCDIC set.

The EBCDIC collating sequence, in ascending order, is:

1. (space)
2. . (period or decimal point)
3. < ("less than" symbol)
4. ((left parenthesis)
5. + (plus sign)
6. \$ (currency symbol)
7. * (asterisk)
8.) (right parenthesis)
9. ; (semicolon)
10. - (hyphen or minus symbol)
11. / (stroke, virgule, slash)
12. , (comma)
13. > ("greater than" symbol)
14. = (equal sign)
15. " (quotation mark)
- 17-42. A thru Z
- 43-52. 0 thru 9

(The complete EBCDIC collating sequence is given in the publication IBM System/360 Reference Data, Order No. X20-1703.)

If one of the operands is described as numeric, it is treated as though it were moved to an alphanumeric data item of the same size and the contents of this alphanumeric data item were then compared to the nonnumeric operand (see "MOVE Statement").

The size of an operand is the total number of characters in the operand.

All group items are treated as nonnumeric operands.

Numeric and nonnumeric operands may be compared only when their USAGE is the same, implicitly or explicitly.



There are two cases of nonnumeric comparison to consider: operands of equal size and operands of unequal size.

Relation Condition

1. Comparison of Operands of Equal Size

Characters in corresponding character positions of the two operands are compared from the high-order end through the low-order end. The high-order end is the leftmost position; the low-order end is the rightmost character position.

If all pairs of characters compare equally through the last pair, the operands are considered equal when the low-order end is reached.

If a pair of unequal characters is encountered, the two characters are compared to determine their relative position in the collating sequence. The operand that contains the character higher in the collating sequence is considered to be the greater operand.

2. Comparison of Operands of Unequal Size

If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by a sufficient number of spaces to make the operands of equal size.

COMPARISONS INVOLVING INDEX-NAMES AND/OR INDEX DATA ITEMS: The comparison of two index-names is equivalent to the comparison of their corresponding occurrence numbers.

In the comparison of an index data item with an index-name or with another index data item, the actual values are compared without conversion.

The comparison of an index-name with a numeric item is permitted if the numeric item is an integer. The numeric integer is treated as an occurrence number. All other comparisons involving an index-name or index data item are not allowed. (For a discussion of indexing, see "Table Handling.")

Permissible comparisons are shown in Figure 23.

First Operand \ Second Operand	GR	AL	AN	ANE	NE	FC* NNL	ZR NL	ED	BI	ID	EF	IF	SR	SN	IN	IDI
Group (GR)	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN		
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Alphanumeric Edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Numeric Edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Figurative Constant (FC)* & Nonnumeric Literal (NNL)	NN	NN	NN	NN	NN			NN			NN		NN	NN		
Fig. Constant ZERO (ZR) & Numeric Literal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU	NU	NN	NU	IO ¹	
External Decimal (ED)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU	IO ¹	
Binary (BI)	NN						NU	NU	NU	NU	NU	NU		NU	IO ¹	
Internal Decimal (ID)	NN						NU	NU	NU	NU	NU	NU		NU	IO ¹	
External Floating Point (EF)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU		
Internal Floating Point (IF)	NN						NU	NU	NU	NU	NU	NU		NU		
Sterling Report (SR)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Sterling Nonreport (SN)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU		
Index Name (IN)								IO ¹	IO ¹	IO ¹	IO ¹				IO	IV
Index Data Item (IDI)															IV	IV

*FC includes all Figurative Constants except ZERO.
¹Valid only if the numeric item is an integer.

NN = comparison as described for nonnumeric operands
 NU = comparison as described for numeric operands
 IO = comparison as described for two index-names
 IV = comparison as described for index data items

Figure 23. Permissible Comparisons

Sign/Switch-status Conditions

Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand (i.e., an item described as numeric) is less than, greater than, or equal to zero.

Format		
{ identifier }	IS [NOT]	{ POSITIVE NEGATIVE ZERO }

An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero. An unsigned field is positive or zero.

Switch-Status Condition

A switch-status condition determines the on or off status of a device switch.

Format
condition-name

The SPECIAL-NAMES paragraph of the Environment Division associates an ON or OFF value (condition-name) with a switch (function-name). The switch-status condition tests the value associated with the switch. The result of the test is true if the switch is set to the position corresponding to condition-name.

COMPOUND CONDITIONS

Two or more simple conditions can be combined to form a compound condition. Each simple condition is separated from the next by one of the logical operators AND or OR.

The logical operators must be preceded by a space and followed by a space. The meaning of the logical operators is as follows:

<u>Logical Operator</u>	<u>Meaning</u>
<u>OR</u>	logical inclusive OR, i.e., either or both are true
<u>AND</u>	logical conjunction, i.e., both are true
<u>NOT</u>	logical negation

Figure 24 shows the relationships between the logical operators and simple conditions A and B, where A and B have the following values:

<u>Values for Condition A</u>	<u>Values for Condition B</u>
True	True
False	True
True	False
False	False

A AND B	A OR B	NOT A	NOT (A AND B)	NOT A AND B	NOT (A OR B)	NOT A OR B
True	True	False	False	False	False	True
False	True	True	True	True	False	True
False	True	False	True	False	False	False
False	False	True	True	False	True	True

Figure 24. Logical Operators and the Resulting Values upon Evaluation

EVALUATION RULES

Logical evaluation begins with the least inclusive pair of parentheses and proceeds to the most inclusive.

If the order of evaluation is not specified by parentheses, the expression is evaluated in the following order:

1. Arithmetic expressions
2. Relational-operators
3. [NOT] condition
4. AND and its surrounding conditions are evaluated first, starting at the left of the expression and proceeding to the right.
5. OR and its surrounding conditions are then evaluated, also proceeding from left to right.

Consider the expression:

A IS NOT GREATER THAN B OR A + B IS EQUAL TO C AND D IS POSITIVE

This will be evaluated as if it were parenthesized as follows:

(A IS NOT GREATER THAN B) OR (((A + B) IS EQUAL TO C) AND (D IS POSITIVE)).

Compound Conditions

The order of evaluation is as follows:

1. (A + B) is evaluated, giving some intermediate result, for example, x.
2. (A IS NOT GREATER THAN B) is evaluated, giving some intermediate truth value, for example, t1.
3. (x IS EQUAL TO C) is evaluated, giving some intermediate truth value, for example, t2.
4. (D IS POSITIVE) is evaluated, giving some intermediate truth value, for example, t3.
5. (t2 AND t3) is evaluated, giving some intermediate truth value, for example, t4.
6. (t1 OR t4) is evaluated, giving the final truth value, and the result of the expression.

Figure 25 shows permissible symbol pairs. A symbol pair in a compound condition is the occurrence of two symbols appearing in sequence.

First Symbol \ Second Symbol	Condition	OR	AND	NOT	()
Condition	-	p	p	-	-	p
OR	p	-	-	p	p	-
AND	p	-	-	p	p	-
NOT	p	-	-	-	p	-
(p	-	-	p	p	-
)	-	p	p	-	-	p

p indicates a permissible pairing
 - indicates that the pairing is not permitted

Figure 25. Permissible Symbol Pairs -- Compound Conditions

IMPLIED SUBJECTS AND RELATIONAL-OPERATORS

When relation conditions are written in a consecutive sequence, any relation condition except the first may be abbreviated by:

1. The omission of the subject of the relation condition, or
2. The omission of the subject and relational-operator of the relation condition.

Within a sequence of relation conditions, both forms of abbreviation may be used. The effect of using such abbreviations is as if the omitted subject was taken from the most recently stated subject, or the omitted relational-operator was taken from the most recently stated relational-operator.

Format of Implied Subject:

...subject relational-operator object

$\left. \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$ [NOT] relational-operator object...

Format of Implied Subject and Relational-operator:

...subject relational-operator object $\left. \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$ [NOT] object...

Ambiguity may result from using NOT in conjunction with abbreviations. When only the subject is implied, NOT is interpreted as a logical operator rather than as part of the relational operator. For example, A NOT > B AND < C is equivalent to A NOT > B AND A < C. When both the subject and the relational operator are implied, NOT is interpreted as part of the relational operator. For example, A NOT > B AND C is equivalent to A NOT > B AND A NOT > C.

The following are examples of implied subjects, and relational-operators. Each example consists of two equivalent statements:

Implied Subject

A = B OR NOT > C (The subject, A, is implied.)
 A = B OR NOT A > C (The subject, A, is explicit.)

Implied Subject and Relational Operator

A = B AND C (Subject and relational-operator, A = , are implied.)
 A = B AND A = C (Subject and relational-operator, A = , are explicit.)
 A NOT = B AND C (Subject and relational-operator, A NOT = , are implied.)
 A NOT = B AND A NOT = C (Subject and relational-operator, A NOT = , are explicit.)

Implied Subject, and Subject and Relational-Operator

A > B AND NOT < C AND D (Subject, A, is implied in the second condition. Subject, A, and relational-operator, <, are implied in the third condition.)
 A > B AND NOT A < C (Subject, A, and relational-operator, <, are explicit.)
 AND A < D

The omitted subject is taken from the most recently stated subject, i.e., A.

The omitted relational-operator is taken from the most recently stated relational-operator, i.e., <.

IF Statement

CONDITIONAL STATEMENTS

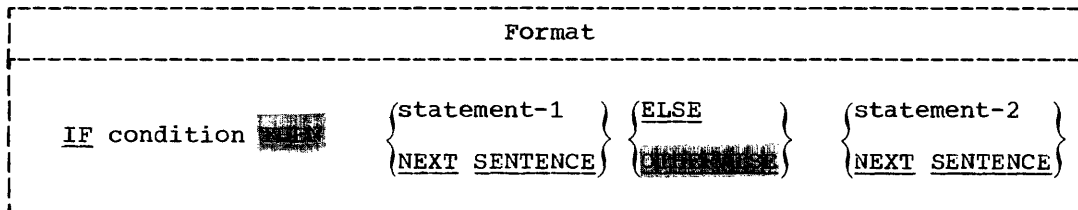
A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value. Conditional statements are listed in "Categories of Statements."

A conditional sentence is a conditional statement optionally preceded by an imperative statement, terminated by a period followed by a space.

Only the IF statement is discussed in this section. Discussion of the other conditional statements is included as part of the description of the associated imperative statements.

IF Statement

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends upon whether the condition is true or false.



The phrase ELSE [REDACTED] NEXT SENTENCE may be omitted if and only if it immediately precedes the period for the sentence.

When an IF statement is executed, the following action is taken:

1. If the condition is true, the statement immediately following the condition or THEN (statement-1) is executed. (If ELSE [REDACTED] is omitted, then all imperative statements following the condition and preceding the period for the sentence are considered to be part of statement-1.) Control is then passed implicitly to the next sentence unless GO TO procedure-name is specified in statement-1. If the condition is true and NEXT SENTENCE is written, control passes explicitly to the next sentence.
2. If the condition is false, either the statement following ELSE [REDACTED] (statement-2) is executed, or, if the ELSE [REDACTED] option is omitted, the next sentence is executed. If the condition is false and NEXT SENTENCE is written following ELSE, control passes explicitly to the next sentence.

When IF statements are not nested, statement-1 and statement-2 must represent imperative statements.

Nested IF Statements

The presence of one or more IF statements within the initial IF statement constitutes a "nested IF statement."

Statement-1 and statement-2 in IF statements may consist of one or more imperative statements and/or a conditional statement. If a conditional statement appears as statement-1 or as part of statement-1, it is said to be nested. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in larger arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already been paired with an ELSE.

In the conditional statement in Figure 26, C stands for condition; S stands for any number of imperative statements; and the pairing of IF and ELSE is shown by the lines connecting them.

Figure 27 is a flowchart indicating the logical flow of the conditional statement in Figure 26.

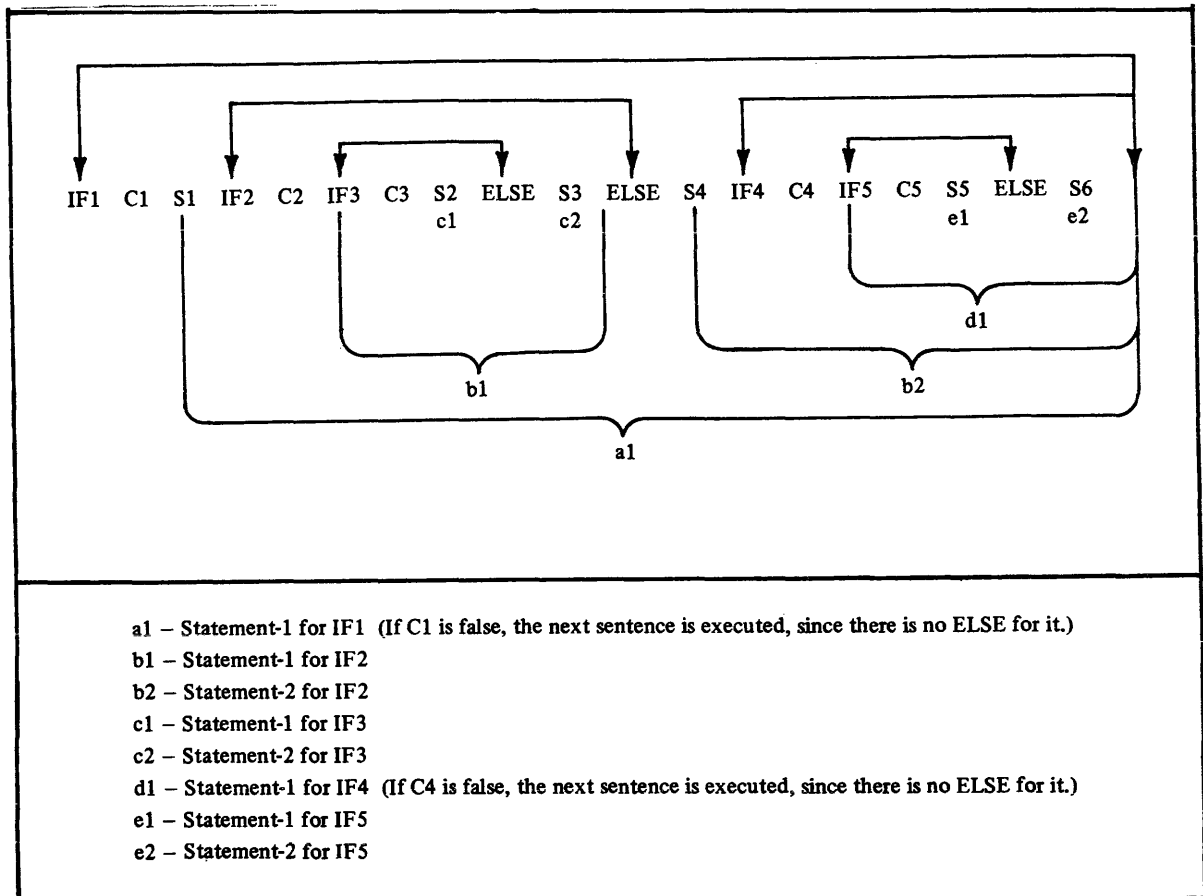


Figure 26. Conditional Statements with Nested IF Statements

IF Statement

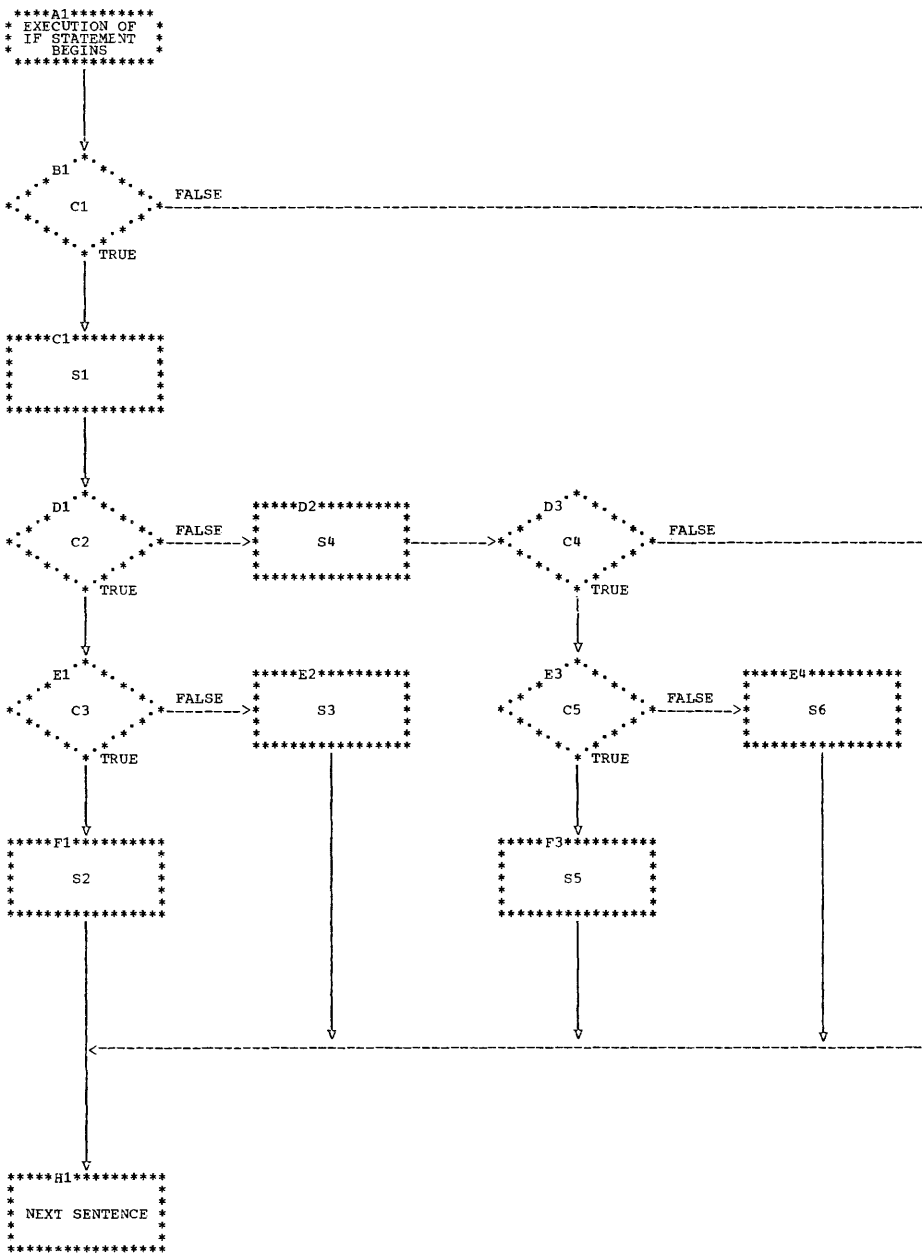


Figure 27. Logical Flow of Conditional Statement with Nested IF Statements

DECLARATIVES

The Declaratives Section provides a method of including procedures that are invoked nonsynchronously: that is, they are executed not as part of the sequential coding written by the programmer, but rather when a condition occurs which cannot normally be tested by the programmer.

Although the system automatically handles checking and creation of standard labels and executes error recovery routines in the case of input/output errors, additional procedures may be specified by the COBOL programmer. The Report Writer feature also uses declarative procedures.

Since these procedures are executed only when labels of a file are to be processed, or at the time an error in reading or writing occurs or when a report group is to be produced, they cannot appear in the regular sequence of procedural statements. They must be written at the beginning of the Procedure Division in a subdivision called DECLARATIVES. A group of declarative procedures constitutes a declarative section. Related procedures are preceded by a USE sentence that specifies their function. A declarative section ends with the occurrence of another section-name with a USE sentence or with the words END DECLARATIVES.

The key words DECLARATIVES and END DECLARATIVES must each begin in Area A. No other text may appear on the same line.

```

                                General Format
-----
PROCEDURE DIVISION.

DECLARATIVES.

{section-name SECTION.  USE sentence.
{paragraph-name.  {sentence} ... } ... } ...

END DECLARATIVES.

```

The USE sentence identifies the type of declarative.

There are three formats of the USE sentence. Each is associated with one of the following types of procedures:

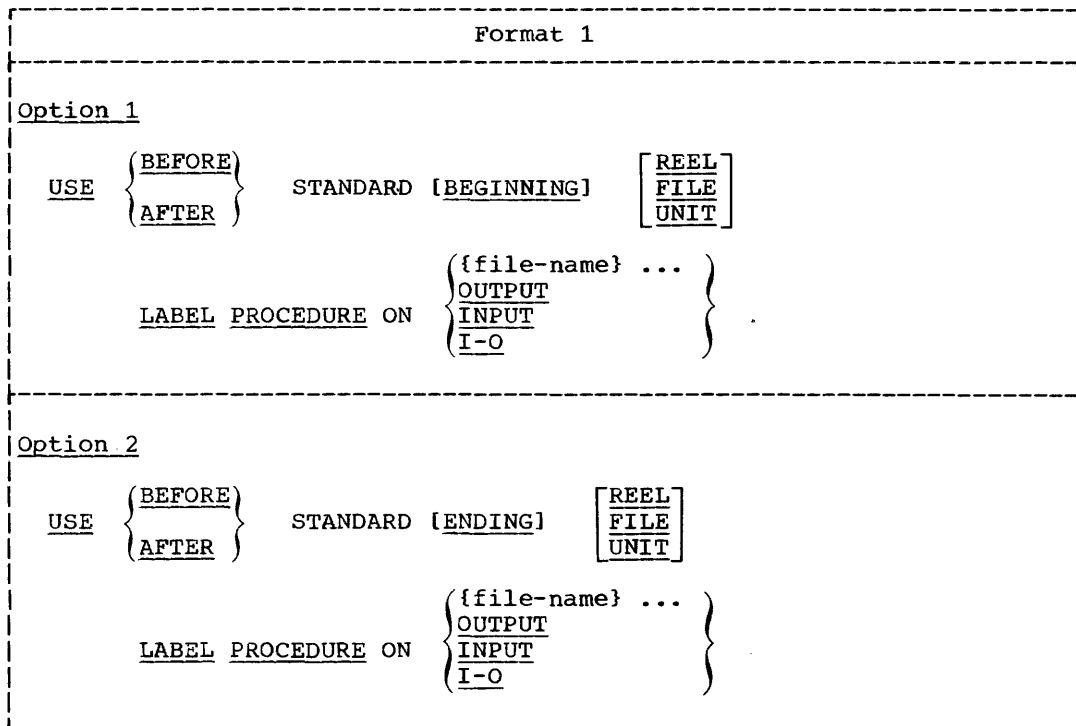
1. Input/output label handling
2. Input/output error-checking procedures
3. Report writing procedures

A USE sentence, when present, must immediately follow a section header in the Declarative portion of the Procedure Division and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

The USE sentence itself is never executed, rather it defines the conditions for the execution of the USE procedure.

LABEL Declarative

Format 1 is used to provide user label-handling procedures. There are two options of Format 1.



When BEFORE is specified, it indicates that nonstandard labels are to be processed. Nonstandard labels may be specified only for tape files.

When AFTER is specified, it indicates that user labels follow standard file labels, and are to be processed.

Note: ASCII considerations for user label-handling procedures are given in Appendix E.

The labels must be listed as data-names in the LABEL RECORDS clause in the File Description entry for the file, and must be described as level-01 data items subordinate to the file entry.

If neither BEGINNING nor ENDING is specified, the designated procedures are executed for both beginning and ending labels.

If UNIT, REEL, or FILE are not included, the designated procedures are executed both for REEL or UNIT, whichever is appropriate, and for FILE labels. The REEL option is not applicable to mass storage files. The UNIT option is not applicable to files in the random access mode since only FILE labels are processed in this mode.

If FILE is specified, the designated procedures are executed at beginning-of-file (on first volume) and/or at end-of-file (on last volume) only. If REEL or UNIT is specified, the designated procedures are executed at beginning-of-volume (on each volume but the first) and/or at end-of-volume (on each volume but the last.) Both BEGINNING and ENDING label processing is executed if BEGINNING or ENDING has not been specified.

The same file-name may appear in different specific arrangements of Format 1. However, appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative.

If the file-name option is used, the File Description entry for file-name must not specify a LABEL RECORDS ARE OMITTED clause.

The user label procedures are executed as follows when the OUTPUT, INPUT, or I-O options are specified:

- When OUTPUT is specified, only for files opened as output.
- When INPUT is specified, only for files opened as input.
- When I-O is specified, only for files opened as I-O.

The file-name must not represent a sort-file.

If the INPUT, OUTPUT, or I-O option is specified, and an input, output, or input-output file, respectively, is described with a LABEL RECORDS ARE OMITTED clause, the USE procedures do not apply.

The standard system procedures are performed:

1. Before or after the user's beginning or ending input label check procedure is executed.
2. Before the user's beginning or ending output label is created.
3. After the user's beginning or ending output label is created, but before it is written on tape.
4. Before or after the user's beginning or ending input-output label check procedure is executed.

Within the procedures of a USE declarative in which the USE sentence specifies an option other than file-name, references to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but does not appear in any data record of this program. Such items must have identical descriptions and positions within each label record.

Within a Format 1 declarative section there must be no reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declaratives section, except that PERFORM statements may refer to a USE procedure, or to procedures associated with it.

The exit from a Format 1 declarative section is inserted by the compiler following the last statement in the section. All logical program paths within the section must lead to the exit point.

There is one exception: a special exit may be specified by the statement GO NO MORE LABELS. When an exit is made from a Format 1 declarative section by means of this statement, the system will do one of the following:

1. Write the current beginning or ending label and then re-enter the USE section at its beginning for further creation of labels. After creating the last label, the user must exit by executing the last statement of the section.
2. Read an additional beginning or ending label, and then re-enter the USE section at its beginning for further checking of labels. When processing user labels, the section will be re-entered only if

there is another user label to check. Hence, there need not be a program path that flows through the last statement in the section. For nonstandard labels, the compiler does not know how many labels exist. Therefore, the last statement in the section must be executed to terminate nonstandard label processing.

If a GO TO MORE-LABELS statement is not executed for a user label, the declarative section is not re-entered to check or create any immediately succeeding user labels.

When reading nonstandard header labels, it is the user's responsibility to read any tape marks that are used to terminate labels. The GO TO MORE-LABELS exit must be used and the declarative must recognize that a tape mark rather than data is being read. The final exit from the declarative must not be taken until the file is positioned just before the first data record.

No tape marks are written following nonstandard header labels. A tape mark is written following the last nonstandard trailer label on each reel.

SAMPLE LABEL DECLARATIVE PROGRAM

The following program creates two files, one with user labels, the other with nonstandard labels. To create the labels, the program contains a DECLARATIVES section, with USE procedures for creating both header and trailer labels.

The program illustrates the following items:

- ① For the two files requiring label creation, the LABEL RECORDS clause specifies the data-name option.
- ② The user labels are created by a USE AFTER BEGINNING/ENDING LABEL procedure.
- ③ Two user header labels are to be created. ~~Therefore the program will create two user header labels.~~
- ④ The nonstandard labels are created by a USE BEFORE BEGINNING/ENDING LABEL procedure.
- ⑤ Label information for the program is taken in part from the input file; therefore input records containing the information must be read and stored before the output files are opened, and the header label procedures invoked.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. LABELPGM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-F50.
OBJECT-COMPUTER. IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT NO-LBL ASSIGN TO SYS010-UT-2400-S.
    SELECT USER ASSIGN TO SYS011-UT-2400-S.
    SELECT NON-STD ASSIGN TO SYS012-UT-2400-S.
DATA DIVISION.
FILE SECTION.
FD NO-LBL
RECORD CONTAINS 80 CHARACTERS
DATA RECORDS ARE IN-REC IN-LBL-HIST
LABEL RECORD IS OMITTED.
01 IN-REC.
    05 TYPEN PIC X(4).
       88 NSTD VALUE 'NSTD'.
    05 DEPT-ID PIC X(11).
    05 BIL-PERIOD PIC X(5).
    05 NAME PIC X(20).
    05 AMOUNT PIC 9(6).
    05 FIL-NAM PIC X(15).
    05 SECUR-CODE PIC XX.
    05 AREAN PIC 9.
       88 HDR-REC VALUE 9.
    05 ACCT-NUM PIC 9(10).
    05 SER-NUM PIC 9(6).
01 IN-LBL-HIST.
    05 FILLER PIC X(4).
    05 FILE-HISTORY PIC X(76).
FD USER
RECORD CONTAINS 80 CHARACTERS
BLOCK CONTAINS 5 RECORDS
DATA RECORD IS USR-REC
LABEL RECORDS ARE USR-LBL USR-LBL-HIST.
① 01 USR-LBL.
    05 USR-HDR PIC X(4).
    05 DEPT-ID PIC X(11).
    05 USR-REC-CNT PIC 9(8) COMP-3.
    05 BIL-PERIOD PIC X(5).
    05 FILLER PIC X(53).
    05 SECUR-CODE PIC XX.
01 USR-LBL-HIST.
    05 FILLER PIC X(4).
    05 LBL-HISTORY PIC X(76).
01 USR-REC.
    05 TYPEN PIC X(4).
    05 FILLER PIC X(5).
    05 NAME PIC X(20).
    05 FILLER PIC X(4).
    05 ACCT-NUM PIC 9(10).
    05 AMOUNT PIC 9(6) COMP-3.
    05 FILLER PIC X(25).
    05 U-SFQ-NUMB PIC 9(8).
FD NON-STD
RECORDING MODE IS U
DATA RECORDS ARE NSTD-REC1 NSTD-REC2
LABEL RECORD IS NSTD-LBL.
① 01 NSTD-LBL.
    05 NSTD-HDR PIC X(7).
    05 NSTD-REC-CNT PIC 9(8) COMP-3.
    05 FILLER PIC X(3).
    05 FIL-NAM PIC X(15).
    05 DEPT-ID PIC X(11).
    05 SER-NUM PIC 9(6).

```

LABEL Declarative -- Sample Program

```
05 CREAT-DATE PIC X(8).
01 NSTD-REC1.
05 ACCT-NUM PIC 9(10).
05 BIL-PERIOD PIC X(5).
05 NAME PIC X(20).
05 FILLER PIC X(40).
05 AREAN PIC 9.
05 FILLER PIC X(20).
05 AMOUNT PIC 9(6) COMP-3.
01 NSTD-REC2.
05 ACCT-NUM PIC 9(10).
05 BIL-PERIOD PIC X(5).
05 NAME PIC X(20).
05 FILLER PIC X(15).
05 DEPT-ID PIC X(11).
05 AMOUNT PIC 9(6).
05 N-SEQ-NUMB PIC 9(8).
05 AREAN PIC 9.
05 FILLER PIC X(4).
WORKING-STORAGE SECTION.
77 N-REC-NUMB PIC 9(8) VALUE ZERO.
77 U-REC-NUMB PIC 9(8) VALUE ZERO.
77 SAV-FIL-NAM PIC X(15).
77 SAV-DEPT-ID PIC X(11).
77 LBL-SWITCH PIC 9 VALUE ZERO.
77 USER-SWITCH PIC 9 VALUE ZERO.
77 NSTD-SWITCH PIC 9 VALUE ZERO.
77 NSTD-REC2-CNT PIC 9(8) VALUE ZERO.
01 STOR-REC.
05 DEPT-ID PIC X(11).
05 BIL-PERIOD PIC X(5).
05 SECUR-CODE PIC XX.
PROCEDURE DIVISION.
```

DECLARATIVES.

- ② USR-HDR-LBL SECTION. USE AFTER BEGINNING FILE
LABEL PROCEDURE ON USER.
A. IF LBL-SWITCH = 0
MOVE SPACES TO USR-LBL
MOVE ZEROES TO USR-REC-CNT
MOVE 'UHL1' TO USR-HDR
MOVE CORRESPONDING STOR-REC TO USR-LBL
ADD 1 TO LBL-SWITCH GO TO MORE-LABELS
ELSE MOVE 'UHL2' TO USR-HDR
MOVE FILE-HISTORY TO LBL-HISTORY.
- ② USR-TRLR-LBL SECTION. USE AFTER ENDING FILE
LABEL PROCEDURE ON USER.
B. MOVE SPACES TO USR-LBL.
MOVE 'UTL1' TO USR-HDR.
MOVE SAV-DEPT-ID TO DEPT-ID IN USR-LBL.
MOVE U-REC-NUMB TO USR-REC-CNT,
- ④ NSTD-HDR-LBL SECTION. USE BEFORE BEGINNING FILE
LABEL PROCEDURE ON NON-STD.
C. MOVE 'NSTHDR1' TO NSTD-HDR.
MOVE ZEROES TO NSTD-REC-CNT
MOVE CORRESPONDING IN-REC TO NSTD-LBL.
MOVE CURRENT-DATE TO CREAT-DATE.
MOVE FIL-NAM OF IN-RFC TO SAV-FIL-NAM.
- ④ NSTD-TRLR-LBL SECTION. USE BEFORE ENDING FILE
LABEL PROCEDURE ON NON-STD.
D. MOVE SPACES TO NSTD-LBL.
MOVE 'NSTEOF ' TO NSTD-HDR.
MOVE N-REC-NUMB TO NSTD-REC-CNT.
MOVE SAV-FIL-NAM TO FIL-NAM IN NSTD-LBL.
END DECLARATIVES.

NON-DECLARATIVE SECTION.

OPEN INPUT NO-LBL.

```

READ-IN.
  READ NO-LBL AT END GO TO END-JOB.
E.  IF NSTD NEXT SENTENCE
    ELSE GO TO PROCESS-USER.
    IF NSTD-SWITCH = 1 NEXT SENTENCE
      ELSE ADD 1 TO NSTD-SWITCH
        OPEN OUTPUT NON-STD
        GO TO READ-IN.
    (5) ADD 1 TO N-REC-NUMB.
    IF HDR-REC MOVE SPACES TO NSTD-REC1
      MOVE CORRESPONDING IN-REC TO NSTD-REC1
      WRITE NSTD-REC1
    ELSE ADD 1 TO NSTD-REC2-CNT
      MOVE SPACES TO NSTD-REC2
      MOVE CORRESPONDING IN-REC TO NSTD-REC2
      MOVE NSTD-REC2-CNT TO N-SEQ-NUMB
      WRITE NSTD-REC2.
    GO TO READ-IN.
PROCESS-USER.
  IF USER-SWITCH = 1 NEXT SENTENCE
    ELSE ADD 1 TO USER-SWITCH
      MOVE CORRESPONDING IN-REC TO STOR-REC
      MOVE DEPT-ID OF IN-REC TO SAV-DEPT-ID
    PERFORM READ-IN
      OPEN OUTPUT USER
      GO TO READ-IN.
    (5) ADD 1 TO U-REC-NUM3.
    MOVE CORRESPONDING IN-REC TO USR-REC.
    MOVE U-REC-NUMB TO U-SEQ-NUMB
    WRITE USR-REC
    GO TO READ-IN.
END-JOB.
  CLOSE NO-LBL.
  IF USER-SWITCH = 1 CLOSE USER.
  IF NSTD-SWITCH = 1 CLOSE NON-STD.
  STOP RUN.

```

A Format 2 USE sentence specifies procedures to be followed if an input/output error occurs during file processing.

Format 2

USE AFTER STANDARD ERROR PROCEDURE

ON { {file-name-1} ...
~~file-name-2 GIVING data-name-1 [data-name-2]~~
INPUT
OUTPUT
I-O }

When Format 2 is used, automatic system error routines are executed before user-specified procedures.

USE declaratives which specify error handling procedures are activated when an input/output error occurs during execution of a READ, WRITE, ~~REWRITE~~, or ~~START~~ statement.

Within the section, the file associated with the USE sentence may not be referred to by an OPEN, ~~START~~, SEEK, READ, WRITE, or ~~REWRITE~~ statement.

ERROR Declarative

Within a USE procedure there must be no reference to nondeclarative procedures except when an exit is taken with a GO TO statement. Conversely, in the nondeclarative portion, there must be no reference to procedure-names that appear in the declaratives portion, except that PERFORM statements may refer to a USE declarative, or to procedures associated with such a declarative.

When either the file-name-1 or file-name-2 option is used, user error handling procedures are executed for input/output errors occurring during execution of a READ, WRITE, REWRITE, or START statement for that file only.

A file-name must not be referred to, implicitly or explicitly, in more than one Format 2 USE sentence.

The user error procedures are executed when the INPUT, OUTPUT, or I-O options are specified and an input/output error occurs, as follows:

- When INPUT is specified, only for files opened as INPUT.
- When OUTPUT is specified, only for files opened as OUTPUT.
- When I-O is specified, only for files opened as I-O (input-output).

When the GIVING option is used, data-name-1 will be set to reflect the error condition that caused the error declarative to be entered. A value of 1 in an error byte indicates that its corresponding error condition has occurred. Individual bytes will correspond to specific error conditions depending on the type of file being processed as shown in Figure 28.

The GIVING option must not be specified for files on unit record devices.

Error Byte	INDEXED SEQUENTIAL	DIRECT	SEQUENTIAL
1	DASD error	Data check in count	Parity error
2	Wrong length record	Wrong length record	Wrong length record
3	Prime data full	No room found	-
4	Cylinder index too small	Data check in key or data	-
5	Master index too small	-	-
6	Overflow area full	-	-
7	No EOF record written in prime data area	-	-
8	-	-	-

Figure 28. Error Byte Meaning for the GIVING Option of an Error Declarative

data-name-1 must be an 8-byte external-decimal item. It must be defined in the Working-Storage Section.

Data-name-2, if specified, will contain the block in error (or blank, if no data was transmitted). Data-name-2 will contain the block in error only when the error occurs during a READ operation.

Data-name-2 must be large enough to hold the largest physical block that exists or will be written on file-name-2. It must be defined in the Working-Storage or Linkage Section. If data-name-2 is defined in the Linkage Section, the block in error is referenced in the buffer area; no storage need be defined for the error block.

An exit from this type of declarative section can be effected by executing the last statement in the section (normal return), or by means of a GO TO statement. Figure 29 summarizes the facilities and limitations associated with each file-processing technique when an error occurs.

The normal return from an error declarative is to the statement following the input/output statement that caused the error.

A Format 3 USE sentence specifies Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced (see "Report Writer").

Format 3

USE BEFORE REPORTING data-name.

ERROR Declarative

Access	Organi- zation	Type of I/O Statement	Error Bytes	Error Declarative Written		No Error Declarative Written
				Normal Return	GO TO Exit	
SEQUEN- TIAL (or not speci- fied)	Sequen- tial	READ	1 or 2	Continued process- ing of file per- mitted; bad block is bypassed	User limited to CLOSE for file	Diagnostic error message is printed; job is terminated
		WRITE	1 or 2	Continued process- ing of file per- mitted; bad block has been written	Continued process- ing of file per- mitted; bad block has been written	
	Direct	READ	1,2, or 4	User limited to CLOSE for file	User limited to CLOSE for file	
RANDOM	Direct	READ	1,2, or 4	Continued process- ing of file per- mitted; bad block has not been by- passed	Continued process- ing of file per- mitted; bad block has not been by- passed	
		WRITE	1,2,3, or 4	Continued process- ing of file per- mitted; bad block has been written	Continued process- ing of file per- mitted; bad block has been written	
		REWRITE	1,2, or 4			
SEQUEN- TIAL (or not speci- fied)	Indexed	READ	1 or 2	Continued process- ing of file per- mitted; bad block has not been by- passed	Continued process- ing of file per- mitted; bad block has not been by- passed	
		START	1	Continued process- ing of file per- mitted	Continued process- ing of file per- mitted	
		REWRITE	1 or 2	Continued process- ing of file per- mitted; bad block has been written	Continued process- ing of file per- mitted; bad block has been written	
		WRITE	1,2,3, 4, or 5	User limited to CLOSE for file	User limited to CLOSE for file	
RANDOM	Indexed	READ	1 or 2	Continued process- ing of file per- mitted; bad block has not been by- passed	Continued process- ing of file per- mitted; bad block has not been by- passed	
		REWRITE	1 or 2	Continued process- ing of file per- mitted; bad block has been written	Continued process- ing of file per- mitted; bad block has been written	
		WRITE	1,2, or 6	User limited to CLOSE for file	User limited to CLOSE for file	

Figure 29. File Processing Techniques and Associated Error Declaratives Capabilities

ARITHMETIC STATEMENTS

The arithmetic statements are used for computations. Individual operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. These operations can be combined symbolically in a formula, using the COMPUTE statement.

Because there are several options common to the arithmetic statements, their discussion precedes individual statement descriptions.

CORRESPONDING Option

The CORRESPONDING option enables computations to be performed on elementary items of the same name simply by specifying the group item to which they belong. The word CORRESPONDING may be abbreviated as CORR.

Both identifiers following CORRESPONDING must refer to group items. For the purposes of this discussion, these identifiers will be called d_1 and d_2 .

Elementary data items from each group are considered CORRESPONDING when both data items have the same name and qualification, up to but not including d_1 and d_2 .

Neither d_1 nor d_2 may be a data item with level number 66, 77, or 88, nor may either be described with the USAGE IS INDEX clause. Neither d_1 nor d_2 may be a FILLER item.

Each data item subordinate to d_1 or d_2 that is described with a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored; any items subordinate to such data items are also ignored. However, d_1 or d_2 may themselves be described with REDEFINES or OCCURS clauses, or be subordinate to items described with REDEFINES or OCCURS clauses.

Each FILLER item subordinate to d_1 or d_2 is ignored. However, d_1 or d_2 may be subordinate to a FILLER item.

GIVING Option

If the GIVING option is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. This identifier, since not itself involved in the computation, may be a numeric edited item.

ROUNDED Option

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

SIZE ERROR Option/Overlapping Operands

When the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless ROUNDED is specified. When ROUNDED is specified, the least significant digit of the resultant identifier has its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing P's and when the number of places in the calculated result exceeds this size, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

Note: The preceding ROUNDED description does not apply when the resultant field is floating point, in which case rounding has no meaning. However, if at least one of the operands of an arithmetic operation is floating point and the resultant field is fixed point, rounding always takes place, whether or not ROUNDED is specified.

SIZE ERROR Option

If, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the associated resultant identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR option is specified.

If the SIZE ERROR option is not specified and a size error condition occurs, the value of the resultant identifier affected may be unpredictable.

If the SIZE ERROR option is specified and a size error condition occurs, the value of the resultant identifier affected by the size error is not altered. After completion of the execution of the arithmetic operation, the imperative statement in the SIZE ERROR option is executed.

Note: In the case of overflow and overflow/overflow, or in division by zero, the size error condition is not checked and the SIZE ERROR option is not executed.

Overlapping Operands

When the sending and receiving operands of an arithmetic statement or a MOVE statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

ADD Statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

```

Format 1
-----
ADD { identifier-1 } [ identifier-2 ] ... TO identifier-m [ROUNDED]
    { literal-1   } [ literal-2   ]
    [ identifier-n [ROUNDED]] ... [ON SIZE ERROR imperative-statement]
  
```

```

Format 2
-----
ADD { identifier-1 } { identifier-2 } [ identifier-3 ] ...
    { literal-1   } { literal-2   } [ literal-3   ]
    GIVING identifier-m [ROUNDED] [ON SIZE ERROR imperative-statement]
  
```

```

Format 3
-----
ADD { CORR } identifier-1 TO identifier-2
    { CORRESPONDING }
    [ROUNDED] [ON SIZE ERROR imperative-statement]
  
```

Format 1 -- the values of the operands preceding the word TO are added together, and the sum is added to the current value of identifier-m (identifier-n), etc. The result is stored in identifier-m (identifier-n), etc.

Format 2 -- when the GIVING option is used, there must be at least two operands preceding the word GIVING. The values of these operands are added together, and the sum is stored as the new value of identifier-m.

In Formats 1 and 2 each identifier must refer to an elementary numeric item, with the exception of identifiers appearing to the right of the word GIVING. These may refer to numeric edited data items.

Each literal must be a numeric literal.

The maximum size of each operand is 18 decimal digits. The maximum size of the resulting sum, after decimal point alignment, is 18 decimal digits.

Format 3 -- when the CORRESPONDING option is used, elementary data items within identifier-1 are added to and stored in corresponding elementary data items within identifier-2. Identifier-1 and identifier-2 must be group items.

COMPUTE Statement

When ON SIZE ERROR is used in conjunction with CORRESPONDING, the size error test is made only after the completion of all the ADD operations. If any of the additions produces a size error condition, the resultant field for that addition remains unchanged, and the imperative statement specified in the SIZE ERROR option is executed.

COMPUTE Statement

The COMPUTE statement assigns to a data item the value of a data item, literal, or arithmetic expression.

Format	
<u>COMPUTE</u> identifier-1 [<u>ROUNDED</u>] =	{ identifier-2 literal-1 arithmetic-expression }
[<u>ON SIZE ERROR</u> imperative-statement]	

Literal-1 must be a numeric literal.

Identifier-2 must refer to an elementary numeric item. Identifier-1 may describe a numeric edited data item.

The identifier-2 and literal-1 options provide a method for setting the value of identifier-1 equal to the value of identifier-2 or literal-1.

The arithmetic-expression option permits the use of a meaningful combination of identifiers, numeric literals, and arithmetic operators. Hence, the user can combine arithmetic operations without the restrictions imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

As in all arithmetic statements, the maximum size of each operand is 18 decimal digits.

DIVIDE Statement

The DIVIDE statement is used to find the quotient resulting from the division of one data item into another data item.

Format 1			
<u>DIVIDE</u>	{ identifier-1 } { literal-1 }	<u>INTO</u> identifier-2	[<u>ROUNDED</u>]
[ON <u>SIZE ERROR</u> imperative-statement]			

Format 2				
<u>DIVIDE</u>	{ identifier-1 } { literal-1 }	{ <u>INTO</u> } { <u>BY</u> }	{ identifier-2 } { literal-2 }	<u>GIVING</u> identifier-3
[<u>ROUNDED</u>] [<u>REMAINDER</u> identifier-4]				
[ON <u>SIZE ERROR</u> imperative-statement]				

When Format 1 is used, the value of identifier-1 (or literal-1) is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by the value of the quotient.

When Format 2 is used, the value of identifier-1 (or literal-1) is divided into or by identifier-2 (or literal-2), the quotient is stored in identifier-3, and the remainder optionally is stored in identifier-4.

A remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. ~~When the remainder option is specified, none of the identifiers may refer to floating point items.~~ If the ROUNDED option is also specified, the quotient is rounded after the remainder is determined.

Each identifier must refer to an elementary numeric item except the identifier following the word GIVING, which may be a numeric edited item.

Each literal must be a numeric literal.

The maximum size of the resulting quotient, after decimal point alignment, is 18 decimal digits. The maximum size of the resulting remainder (if specified), after decimal point alignment, is 18 decimal digits.

Division by zero always results in a size error condition.

MULTIPLY Statement

MULTIPLY Statement

The MULTIPLY statement is used to multiply one data item by another data item.

Format 1

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
         { literal-1 }
[ON SIZE ERROR imperative-statement]
```

Format 2

```
MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3
         { literal-1 }     { literal-2 }
[ROUNDED] [ON SIZE ERROR imperative-statement]
```

When Format 1 is used, the value of identifier-1 (or literal-1) is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by the product.

When Format 2 is used, the value of identifier-1 (or literal-1) is multiplied by identifier-2 (or literal-2), and the product is stored in identifier-3.

Each identifier must refer to an elementary numeric item except the identifier following the word GIVING, which may be a numeric edited item.

Each literal must be a numeric literal.

The maximum size of each operand is 18 decimal digits. The maximum size of the resulting product, after decimal point alignment, is 18 decimal digits.

SUBTRACT Statement

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from another data item(s).

```

Format 1
-----
SUBTRACT  {identifier-1}  [identifier-2]  ...
          {literal-1   }  [literal-2   ]
          FROM identifier-m  [ROUNDED]
          [identifier-n [ROUNDED]] ... [ON SIZE ERROR imperative-statement]
  
```

```

Format 2
-----
SUBTRACT  {identifier-1}  [identifier-2]  ...
          {literal-1   }  [literal-2   ]
          FROM  {identifier-m}  GIVING identifier-n
          {literal-m   }
          [ROUNDED]  [ON SIZE ERROR imperative-statement]
  
```

```

Format 3
-----
SUBTRACT  {CORR           }  identifier-1 FROM identifier-2
          {CORRESPONDING}
          [ROUNDED] [ON SIZE ERROR imperative-statement]
  
```

Format 1 -- all literals or identifiers preceding the word FROM are added together, and this total is subtracted from identifier-m and identifier-n (if stated), etc. The result of the subtraction is stored as the new value of identifier-m, identifier-n, etc.

Format-2 -- all literals or identifiers preceding the word FROM are added together, and this total is subtracted from literal-m or identifier-m. The result of the subtraction is stored as the value of identifier-n.

Format 3 -- data items in identifier-1 are subtracted from, and the difference stored into corresponding data items in identifier-2. When the CORRESPONDING option is used in conjunction with ON SIZE ERROR and an ON SIZE ERROR condition arises, the result for SUBTRACT is analogous to that for ADD.

SUBTRACT Statement

Each identifier must refer to an elementary numeric item except the identifier following the word GIVING, which may be a numeric edited item.

Each literal must be a numeric literal.

The maximum size of each operand is 18 decimal digits. The maximum size of the resulting difference, after decimal point alignment, is 18 decimal digits.

PROCEDURE BRANCHING STATEMENTS

Statements, sentences, and paragraphs in the Procedure Division are ordinarily executed sequentially. The procedure branching statements (GO TO, ALTER, PERFORM, STOP, and EXIT) allow alterations in the sequence.

GO TO Statement

The GO TO statement allows a transfer from one part of the program to another.

Format 1
<u>GO TO</u> procedure-name-1

Format 2
<u>GO TO</u> procedure-name-1 [procedure-name-2] ... <u>DEPENDING ON</u> identifier

Format 3
<u>GO TO</u> .

When Format 1 is specified, control is passed to procedure-name-1 or to another procedure name if the GO TO statement has been changed by an ALTER statement. (If the latter is the case, the GO TO statement must have a paragraph name, and the GO TO statement must be the only statement in the paragraph.)

If a GO TO statement represented by Format 1 appears in an imperative sentence, it must appear as the only or last statement in a sequence of imperative statements.

When Format 2 is used, control is transferred to one of a series of procedures, depending on the value of the identifier. When identifier has a value of 1, control is passed to procedure-name-1; a value of 2 causes control to be passed to procedure-name-2, ...; a value of n causes control to be passed to procedure-name-n. For the GO TO statement to have effect, identifier must represent a positive or unsigned integer, i.e., 1, 2, ..., n. If the value of the identifier is anything other than a value within the range 1 through n, the GO TO statement is ignored. The number of procedure-names must not exceed 2031.

ALTER Statement

Identifier is the name of a numeric elementary item described as an integer. Its PICTURE must be of four digits or less. Its USAGE must be DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3.

When Format 3 is used, an ALTER statement, referring to the GO TO statement, must have been executed prior to the execution of the GO TO statement. The GO TO statement must immediately follow a paragraph name and must be the only statement in the paragraph.

ALTER Statement

The ALTER statement is used to change the transfer point specified in a GO TO statement.

Format
<pre>ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2 [procedure-name-3 TO [PROCEED TO] procedure-name-4]...</pre>

Procedure-name-1, procedure-name-3, etc., must be the names of paragraphs that contain only one sentence consisting of a GO TO statement without the DEPENDING option.

Procedure-name-2, procedure-name-4, etc., must be the names of paragraphs or sections in the Procedure Division.

The effect of the ALTER statement is to replace the procedure-name operands of the GO TO statements with procedure-name-2, procedure-name-4, etc., of the ALTER statement, where the paragraph-name containing the GO TO statement is procedure-name-1, procedure-name-3, etc. For example:

```
PARAGRAPH-1.
  GO TO BYPASS-PARAGRAPH.
PARAGRAPH-1A.
.
.
BYPASS-PARAGRAPH.
.
.
  ALTER PARAGRAPH-1 TO PROCEED TO PARAGRAPH-2.
.
.
PARAGRAPH-2.
.
.
```

Before the ALTER statement is executed, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After execution of the ALTER statement, however, when control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Segmentation Information: A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority. All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in an overlayable fixed segment (see "Segmentation").

PERFORM Statement

The PERFORM statement is used to depart from the normal sequence of procedures in order to execute a statement, or a series of statements, a specified number of times; or until a predetermined condition is satisfied. After the statements are executed, control is returned to the statement after the PERFORM statement.

Format 1

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

Format 2

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

```
  {identifier-1}
  {integer-1}   TIMES
```

Format 3

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

```
  UNTIL condition-1
```

Format 4

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

```
  VARYING {index-name-1} FROM {index-name-2}
           {identifier-1} {literal-2}
           {identifier-2}
```

```
  BY {literal-3}
      {identifier-3} UNTIL condition-1
```

```
  [AFTER {index-name-4} FROM {index-name-5}
      {identifier-4} {literal-5}
      {identifier-5}]
```

```
  BY {literal-6}
      {identifier-6} UNTIL condition-2
```

```
  [AFTER {index-name-7} FROM {index-name-8}
      {identifier-7} {literal-8}
      {identifier-8}]
```

```
  BY {literal-9}
      {identifier-9} UNTIL condition-3]]
```

PERFORM Statement

Each procedure-name must be the name of a section or paragraph in the Procedure Division.

Each identifier represents a numeric elementary item described in the Data Division. In Format 2, and Format 4 with the AFTER option, each identifier represents a numeric item described as an integer. However, when Format 4 with the AFTER option is used, this compiler allows each identifier to be described as a non-integer numeric item.

Each literal represents a numeric literal.

Whenever a PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. Control is always returned to the statement following the PERFORM statement. The point from which this control is passed is determined as follows:

1. If procedure-name-1 is a paragraph name and procedure-name-2 is not specified, the return is made after the execution of the last statement of procedure-name-1.
2. If procedure-name-1 is a section name and procedure-name-2 is not specified, the return is made after the execution of the last sentence of the last paragraph in procedure-name-1.
3. If procedure-name-2 is specified and it is a paragraph name, the return is made after the execution of the last statement of that paragraph.
4. If procedure-name-2 is specified and it is a section name, the return is made after the execution of the last sentence of the last paragraph in the section.

When both procedure-name-1 and procedure-name-2 are specified, GO TO and PERFORM statements may appear within the sequence of statements within these paragraphs or sections. When procedure-name-1 alone is specified, PERFORM statements may appear within the procedure. GO TO may also appear but may not refer to a procedure-name outside the range of procedure-name-1.

When a PERFORM statement includes within its range of procedures another PERFORM statement, this embedded PERFORM statement must have its range of procedures either totally included in or totally excluded from the range of procedures of the original PERFORM statement. That is, the exit point of the original PERFORM statement cannot be contained within the range of procedures of the embedded PERFORM statement, except as a common exit point. Embedded PERFORM or GO TO statements may have their exit point at the same point that the original PERFORM makes its exit point. Exit points may be the name of a paragraph, section, or sentence.

Control may be passed to a sequence of statements that lies between the entry and exit points of a PERFORM statement by means other than a PERFORM. In this case, control passes through the last statement of the procedure to the following statement as if no PERFORM statement referred to these procedures.

FORMAT 1: When Format 1 is used, the procedure(s) referred to are executed once, and control returns to the statement following the PERFORM statement.

FORMAT 2: When Format 2 is used, the procedure(s) are performed the number of times specified by identifier-1 or integer-1. Once the TIMES option is satisfied, control is transferred to the statement following the PERFORM statement.

The following rules apply to the use of a Format 2 PERFORM statement:

1. If integer-1 or identifier-1 is zero or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.
2. Once the PERFORM statement has been initiated, any reference to identifier-1 has no effect in varying the number of times the procedures are initiated.

FORMAT 3: When Format 3 is used, the specified procedures are performed until the condition specified by the UNTIL option is true. At this time, control is transferred to the statement following the PERFORM statement. If the condition is true at the time that the PERFORM statement is encountered, the specified procedure(s) are not executed.

FORMAT 4: Format 4 is used to augment the value of one or more identifiers or index-names during the execution of a PERFORM statement.

When executing a Format 4 PERFORM statement, the initial values of identifier-2 (index-name-2) and identifier-5 (index-name-5) must be positive in order to conform with the standard. However, this compiler allows these initial values to be negative.

In the following discussion of Format 4, every reference to identifier-n also refers to index-name-n except when identifier-n is the object of the BY option. Also, when index-names are used, the FROM and BY clauses have the same effect as in a SET statement (see "Table Handling").

During execution of the PERFORM statement, reference to index-names or identifiers of the FROM option has no effect in altering the number of times the procedures are to be executed. Changing the value of index-names or identifiers of the VARYING option or identifiers of the BY option, however, will change the number of times the procedures are executed.

When one identifier is varied, the following is the sequence of events:

1. Identifier-1 is set equal to its starting value, identifier-2 or literal-2.
2. If condition-1 is false, the specified procedure(s) are executed once.
3. The value of identifier-1 is augmented by the specified increment or decrement, identifier-3 or literal-3, and condition-1 is evaluated again.
4. Steps 2 and 3 are repeated, if necessary, until the condition is true. When the condition is true, control passes directly to the statement following the PERFORM statement. If the condition is true for the starting value of identifier-1, the procedure(s) are not executed, and control passes directly to the statement following the PERFORM statement.

Figure 30 is a flowchart illustrating the logic of the PERFORM statement when one identifier is varied.

When two identifiers are varied, the following is the sequence of events:

1. Identifier-1 and identifier-4 are set to their initial values, identifier-2 (or literal-2) and identifier-5 (or literal-5), respectively.
2. Condition-1 is evaluated; if true, control is passed to the statement following the PERFORM statement; if false, condition-2 is evaluated.

PERFORM Statement

3. If condition-2 is false, procedure-name-1 through procedure-name-2 (if specified) is executed once.
4. Identifier-4 is augmented by identifier-6 (or literal-6), and condition-2 is evaluated again.
5. If condition-2 is false, steps 3 and 4 are repeated.
6. If condition-2 is true, identifier-4 is set to its initial value, identifier-5.
7. Identifier-1 is augmented by identifier-3 (or literal-3).
8. Steps 2 through 7 are repeated until condition-1 is true.

At the termination of the PERFORM statement, if condition-1 was true when the PERFORM statement was encountered, identifier-1 and identifier-4 contain their initial values. Otherwise, identifier-1 has a value that differs from its last used setting by an increment or decrement, as the case may be.

Figure 31 is a flowchart illustrating the logic of the PERFORM statement when two identifiers are varied.

For three identifiers, the mechanism is the same as for two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

Figure 32 is a flowchart illustrating the logic of the PERFORM statement when three identifiers are varied.

SEGMENTATION INFORMATION: A PERFORM statement appearing in a section whose priority is less than the segment limit can have within its range only one of the following:

1. Sections each of which has a priority number less than 50
2. Sections wholly contained in a single segment whose priority number is greater than 49

However, this compiler allows the PERFORM to have within its range sections with any priority numbers.

A PERFORM statement appearing in a section whose priority number is equal to or greater than the segment limit can have within its range only one of the following:

1. Sections each of which has the same priority number as that containing the PERFORM statement
2. Sections with a priority number less than the segment limit

However, this compiler allows the PERFORM to have within its range sections with any priority numbers.

When a procedure-name in a permanent segment is referred to by a PERFORM statement in an independent segment, the independent segment is reinitialized upon exit from the performed procedure (see "Segmentation").

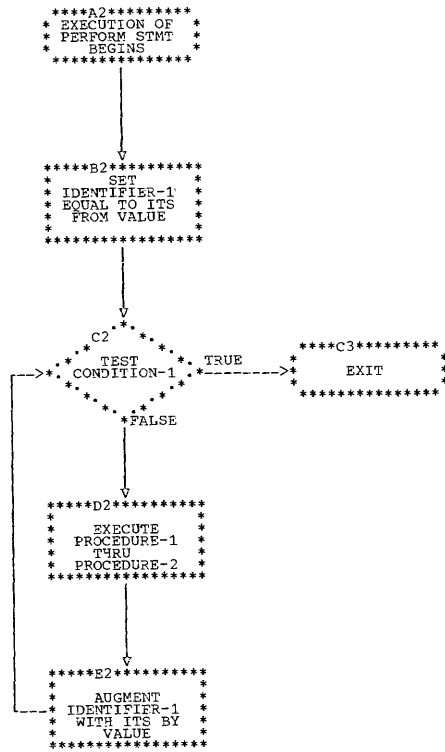


Figure 30. Logical Flow of Option 4 PERFORM Statement Varying One Identifier

PERFORM Statement

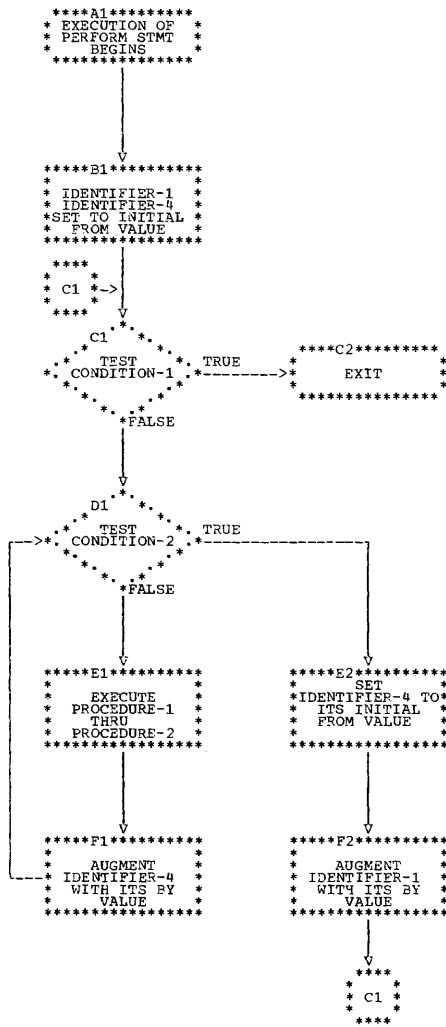


Figure 31. Logical Flow of Option 4 PERFORM Statement Varying Two Identifiers

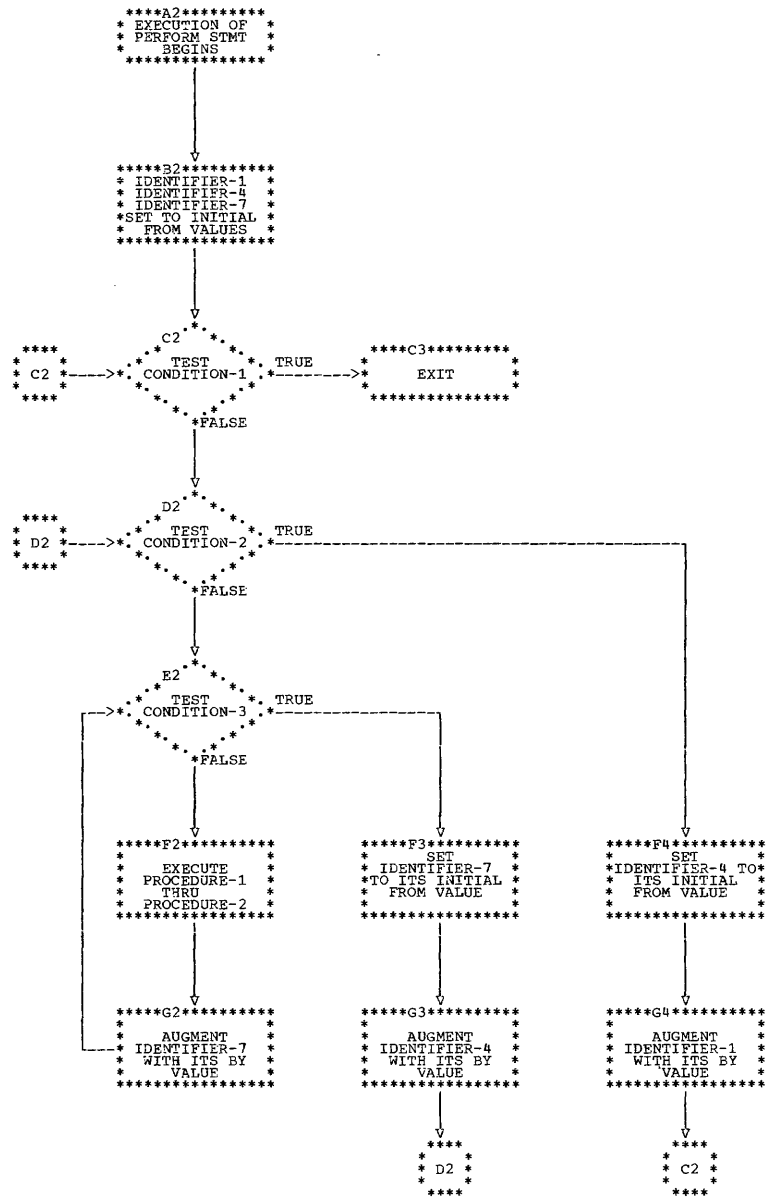


Figure 32. Logical Flow of Option 4 PERFORM Statement Varying Three Identifiers

STOP/EXIT Statements

STOP Statement

The STOP statement halts the object program either permanently or temporarily.

Format	
<u>STOP</u>	{ <u>RUN</u> literal }

When the RUN option is used, the execution of the object program is terminated, and control is returned to the system.

If a STOP statement with the RUN option appears in an imperative statement, it must appear as the only or last statement in a sequence of imperative statements. All files should be closed before a STOP RUN statement is issued.

For the effect when STOP RUN is used in either a calling program or a called program, see "Subprogram Linkage."

When the literal option is used, the literal is communicated to the operator. The program may be resumed only by operator intervention. Continuation of the object program begins with the execution of the next statement in sequence.

The literal may be numeric or nonnumeric, or it may be any figurative constant except ALL.

EXIT Statement

The EXIT statement provides a common end point for a series of procedures.

Format	
paragraph-name.	<u>EXIT</u> [<u>PROGRAM</u>].

It is sometimes necessary to transfer control to the end point of a series of procedures. This is normally done by transferring control to the next paragraph or section, but in some cases this does not have the required effect. For instance, the point to which control is to be transferred may be at the end of a range of procedures governed by a

PERFORM or at the end of a declarative section. The EXIT statement is provided to enable a procedure-name to be associated with such a point.

If control reaches an EXIT paragraph and no associated PERFORM or USE statement is active, control passes through the EXIT point to the first sentence of the next paragraph.

The EXIT statement must be preceded by a paragraph-name and be the only statement in the paragraph.

The EXIT statement with the PROGRAM option is discussed in "Subprogram Linkage."

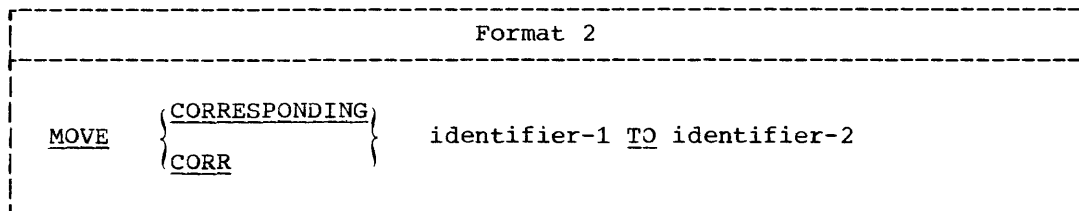
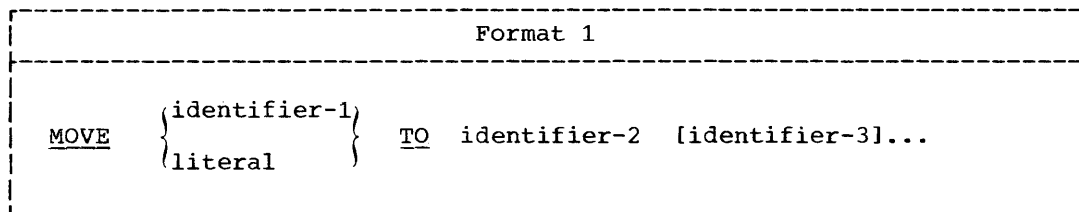
MOVE Statement

DATA-MANIPULATION STATEMENTS

Movement and inspection of data are implicit in the functioning of several of the COBOL statements. These statements are: MOVE, EXAMINE, and TRANSFORM.

MOVE Statement

The MOVE statement is used to transfer data from one area of storage to one or more other areas.



FORMAT 1: identifier-1 and literal represent the sending area; identifier-2, identifier-3, ... represent the receiving areas.

The data designated by literal or identifier-1 is moved first to identifier-2, then to identifier-3 (if specified), etc.

An index data item cannot appear as an operand of a MOVE statement.

FORMAT 2: the CORRESPONDING option is used to transfer data between items of the same name simply by specifying the group items to which they belong.

Neither identifier may be a level-66, level-77, or level-88 data item.

Data items from each group are considered CORRESPONDING when they have the same name and qualification, up to but not including identifier-1 and identifier-2.

At least one of the data items of a pair of matching items must be an elementary data item.

Each subordinate item containing an OCCURS, REDEFINES, USAGE IS INDEX, or RENAMES clause is ignored. However, either identifier may have a REDEFINES or OCCURS clause in its description or may be subordinate to a data item described with these clauses.

General Rules Applying to Any MOVE Statement:

1. Any move in which the sending and receiving items are both elementary items is an elementary move. Each elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, or alphanumeric edited (see "PICTURE Clause" in "Data Division"). Numeric literals belong to the category numeric; nonnumeric literals belong to the category alphanumeric.
2. When an alphanumeric edited, alphanumeric, or alphabetic item is a receiving item:
 - a. Justification and any necessary filling of unused character positions takes place as defined under the JUSTIFIED clause. Unused character positions are filled with spaces.
 - b. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated after the receiving item is filled.
 - c. If the sending item has an operational sign, the absolute value is used.
3. When a numeric or numeric edited item is a receiving item:
 - a. Alignment by decimal point and any necessary zero filling of unused character positions takes place, except when zeros are replaced because of editing requirements.
 - b. The absolute value of the sending item is used if the receiving item has no operational sign.
 - c. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, excess digits are truncated.
 - d. The results at object time may be unpredictable if the sending item contains any nonnumeric characters.
4. Any necessary conversion of data from one form of internal representation to another takes place during the move, along with any specified editing in the receiving item.
5. Any move that is not an elementary move is treated exactly as though it were an alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area is filled without consideration for the individual elementary or group items contained within either the sending or the receiving area.
6. When the sending and receiving operands of a MOVE statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

There are certain restrictions on elementary moves. These are shown in Figure 33.

MOVE Statement

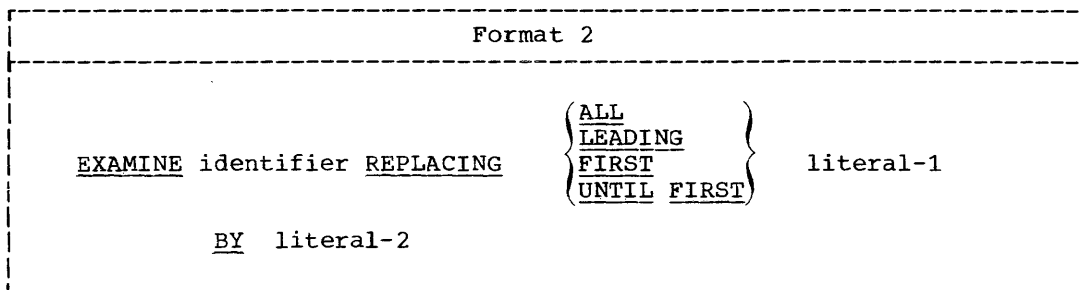
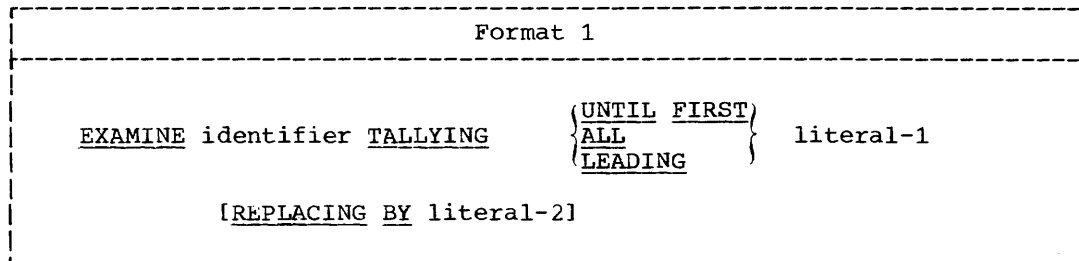
Source Field	Receiving Field												
	GR	AL	AN	ED	BI	NE	ANE	ID	EF	IF	SN	SR	
Group (GR)	Y	Y	Y	Y ¹	Y ¹	Y ¹	Y ¹	Y ¹	Y ¹	Y ¹	Y ¹	Y ¹	
Alphabetic (AL)	Y	Y	Y	N	N	N	Y	N	N	N	N	N	
Alphanumeric (AN)	Y	Y	Y	Y ⁴	Y ⁴	Y ⁴	Y	Y ⁴	Y ⁴	Y ⁴	Y ⁴	Y ⁴	
External Decimal (ED)	Y ¹	N	Y ²	Y	Y	Y	Y ²	Y	Y	Y	Y	Y	
Binary (BI)	Y ¹	N	Y ²	Y	Y	Y	Y ²	Y	Y	Y	Y	Y	
Numeric Edited (NE)	Y	N	Y	N	N	N	Y	N	N	N	N	N	
Alphanumeric Edited (ANE)	Y	Y	Y	N	N	N	Y	N	N	N	N	N	
ZEROS (numeric or alphanumeric)	Y	N	Y	Y ³	Y ³	Y ³	Y	Y ³	Y ³	Y ³	Y ³	Y ³	
SPACES (AL)	Y	Y	Y	N	N	N	Y	N	N	N	N	N	
HIGH-VALUE, LOW-VALUE, QUOTES	Y	N	Y	N	N	N	Y	N	N	N	N	N	
ALL literal	Y	Y	Y	Y ⁵	Y ⁵	Y ⁵	Y	Y ⁵	N	N	N	N	
Numeric Literal	Y ¹	N	Y ²	Y	Y	Y	Y ²	Y	Y	Y	Y	Y	
Nonnumeric Literal	Y	Y	Y	Y ⁵	Y ⁵	Y ⁵	Y	Y ⁵	N	N	N	N	
Internal Decimal (ID)	Y ¹	N	Y ²	Y	Y	Y	Y ²	Y	Y	Y	Y	Y	
External Floating-point (EF)	Y ¹	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	
Internal Floating-point (IF)	Y ¹	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	
Sterling Nonreport (SN)	Y ¹	N	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	
Sterling Report (SR)	Y	N	Y	N	N	N	Y	N	N	N	N	N	
Floating-point Literal	Y ¹	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	

¹Move without conversion (like AN to AN)
²Only if the decimal point is at the right of the least significant digit
³Numeric move
⁴The alphanumeric field is treated as an ED (integer) field
⁵The literal must consist only of numeric characters and is treated as an ED integer field

Figure 33. Permissible Moves

EXAMINE Statement

The EXAMINE statement is used to count the number of times a specified character appears in a data item and/or to replace a character with another character.



In all cases, the description of identifier must be such that its usage is display (explicitly or implicitly).

When identifier represents a nonnumeric data item, examination starts at the leftmost character and proceeds to the right. Each character in the data item is examined in turn. For purposes of the EXAMINE statement, external floating-point items are treated as nonnumeric data items.

When identifier represents a numeric data item, this data item must consist of numeric characters, and may possess an operational sign. Examination starts at the leftmost character and proceeds to the right. Each character is examined in turn.

If the letter 'S' is used in the PICTURE of the data item description to indicate the presence of an operational sign, the sign is ignored by the EXAMINE statement.

Each literal must consist of a single character belonging to a class consistent with that of the identifier; in addition, each literal may be any figurative constant except ALL. If identifier is numeric, each literal must be an unsigned integer or the figurative constant ZERO (ZEROES, ZEROS).

When Format 1 is used, an integral count is created which replaces the value of a special register called TALLY, whose implicit description is that of an unsigned integer of five digits (see "Language Considerations").

EXAMINE Statement

1. When the ALL option is used, this count represents the number of occurrences of literal-1.
2. When the LEADING option is used, this count represents the number of occurrences of literal-1 prior to encountering a character other than literal-1.
3. When the UNTIL FIRST option is used, this count represents all characters encountered before the first occurrence of literal-1.

Whether Format 2 is used, or the REPLACING option of Format 1, the replacement rules are the same. They are as follows:

1. When the ALL option is used, literal-2 is substituted for each occurrence of literal-1.
2. When the LEADING option is used, the substitution of literal-2 for each occurrence of literal-1 terminates as soon as a character other than literal-1 or the right-hand boundary of the data item is encountered.
3. When the UNTIL FIRST option is used, the substitution of literal-2 terminates as soon as literal-1 or the right-hand boundary of the data item is encountered.
4. When the FIRST option is used, the first occurrence of literal-1 is replaced by literal-2.

Specific EXAMINE statements showing the effect of each statement on the associated data item and the TALLY are shown in Figure 34.

EXAMINE Statement	ITEM-1 (Before)	Data (After)	Result- ing Value of TALLY
EXAMINE ITEM-1 TALLYING ALL 0	101010	101010	3
EXAMINE ITEM-1 TALLYING ALL 1 REPLACING BY 0	101010	000000	3
EXAMINE ITEM-1 REPLACING LEADING "*" BY SPACE	**7000	7000	†
EXAMINE ITEM-1 REPLACING FIRST "*" by "\$"	**1.94	\$*1.94	†
† unchanged			

Figure 34. Examples of Data Examination

TRANSFORM Statement

The TRANSFORM statement is used to alter characters according to a transformation rule.

Format			
TRANSFORM	identifier-3	CHARACTERS FROM	{ figurative-constant-1 nonnumeric-literal-1 identifier-1 }
TO	{ figurative-constant-2 nonnumeric-literal-2 identifier-2 }		

Identifier-3 must represent an elementary alphabetic, alphanumeric, or numeric edited item, or a group item.

The combination of the FROM and TO options determines what the transformation rule is.

The following rules pertain to the operands of the FROM and TO options:

1. Nonnumeric literals require enclosing quotation marks.
2. Identifier-1 and identifier-2 must be elementary alphabetic, or alphanumeric items, or fixed length group items not over 255 characters in length.
3. A character may not be repeated in nonnumeric-literal-1 or in the area defined by identifier-1. If a character is repeated, the results will be unpredictable.
4. The allowable figurative constants are: ZERO, ZEROES, ZEROS, SPACE, SPACES, QUOTE, QUOTES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, and LOW-VALUES.

When either identifier-1 or identifier-2 appears as an operand of the specific transformation, the user can change the transformation rule at object time.

Examples of data transformation are given in Figure 35; combinations of the FROM and TO options are shown in Figure 36.

If any of the operands of a TRANSFORM statement share a part of their storage (that is, if the operands overlap), the result of the execution of such a statement is unpredictable.

Identifier-3 (Before)	FROM	TO	Identifier-3 (After)
1b7bbABC	SPACE	QUOTE	1"7""ABC
1b7bbABC	"17CB"	"QRST"	QbRbbATS
1b7bbABC	b17ABC	CBA71b	BCACC71b
1234WXY89	98YXW4321	ABCDEFGHI	IHG FEDCBA

Figure 35. Examples of Data Transformation

TRANSFORM Statement

Operands	Transformation Rule
FROM figurative-constant-1 TO figurative-constant-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character figurative-constant-2.
FROM figurative-constant-1 TO nonnumeric-literal-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character nonnumeric-literal-2.
FROM figurative-constant-1 TO identifier-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character represented by identifier-2.
FROM nonnumeric-literal-1 TO figurative-constant-2	All characters in the data item represented by identifier-3 that are equal to any character in nonnumeric-literal-1 are replaced by the single character figurative-constant-2.
FROM nonnumeric-literal-1 TO nonnumeric-literal-2	<p>Nonnumeric-literal-1 and nonnumeric-literal-2 must be equal in length or nonnumeric-literal-2 must be a single character.</p> <p>If the nonnumeric-literals are equal in length, any character in the data item represented by identifier-3 equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of nonnumeric-literal-2.</p> <p>If the length of nonnumeric-literal-2 is one, all characters in the data item represented by identifier-3 that are equal to any character appearing in nonnumeric-literal-1 are replaced by the single character given in nonnumeric-literal-2.</p>

Figure 36.7 Combinations of FROM and TO Options (Part 1 of 2)

Operands	Transformation Rule
<p>FROM nonnumeric-literal-1 TO identifier-2</p>	<p>Nonnumeric-literal-1 and the data item represented by identifier-2 must be equal in length or identifier-2 must represent a single character item.</p> <p>If nonnumeric-literal-1 and identifier-2 are equal in length, any character represented by identifier-3 equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of the item represented by identifier-2.</p> <p>If the length of the data item represented by identifier-2 is one, all characters represented by identifier-3 that are equal to any character appearing in nonnumeric-literal-1 are replaced by the single character represented by identifier-2.</p>
<p>FROM identifier-1 TO figurative-constant-2</p>	<p>All characters represented by identifier-3 that are equal to any character in the data item represented by identifier-1 are replaced by the single character figurative-constant-2.</p>
<p>FROM identifier-1 TO nonnumeric-literal-2</p>	<p>The data item represented by identifier-1 and nonnumeric-literal-2 must be of equal length or nonnumeric-literal-2 must be one character.</p> <p>If identifier-1 and nonnumeric-literal-2 are equal in length, any character in identifier-3 equal to a character in identifier-1 is replaced by the character in the corresponding position of nonnumeric-literal-2.</p> <p>If the length of nonnumeric-literal-2 is one, all characters represented by identifier-3 that are equal to any character represented by identifier-1 are replaced by the single character given in nonnumeric-literal-2.</p>
<p>FROM identifier-1 TO identifier-2</p>	<p>Any character in the data item represented by identifier-3 equal to a character in the data item represented by identifier-1 is replaced by the character in the corresponding position of the data item represented by identifier-2. Identifier-1 and identifier-2 can be one or more characters, but must be equal in length.</p>

Figure 36. Combinations of FROM and TO Options (Part 2 of 2)

OPEN Statement

INPUT/OUTPUT STATEMENTS

The flow of data through the computer is governed by the Disk Operating System. The COBOL statements discussed in this section are used to initiate the flow of data to and from files stored on external media and to govern low-volume information that is to be obtained from or sent to input/output devices such as a card reader or console typewriter.

The Disk Operating System is a record processing system. That is, the unit of data made available by a READ or passed along by a WRITE is the record. The COBOL user need be concerned only with the use of individual records; provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (where feasible), unblocking and blocking, and volume switching procedures.

Discussions in this section use the terms volume and reel. The term volume applies to all input/output devices. The term reel applies only to tape devices. Treatment of mass storage devices in the sequential access mode is logically equivalent to the treatment of tape files.

Note: The WRITE statement with the BEFORE/AFTER ADVANCING option is referred to in some of the discussions that follow as the WRITE BEFORE/AFTER ADVANCING statement. Similarly, the WRITE statement with the AFTER POSITIONING option is referred to in some of the discussions as the WRITE AFTER POSITIONING statement.

OPEN Statement

The OPEN statement initiates the processing of input, output, and input-output files. It performs checking and/or writing of labels and other input/output operations.

```
Format
-----
OPEN [INPUT {file-name [REVERSED] }...]
      [WITH NO REWIND]
      [OUTPUT {file-name [WITH NO REWIND] }...]
      [I-O {file-name} ...]
```

At least one of the options INPUT, OUTPUT, or I-O must be specified. However, there must be no more than one instance of each option in the same statement, although more than one file-name may be used with each option. These options may appear in any order.

The I-O option pertains only to mass storage files.

The file-name must be defined by a file description entry in the Data Division. The FD entry for the file must be equivalent to that specified when the file was created.

The OPEN statement must not specify a sort-file; an OPEN statement must be specified for all other files. The OPEN statement for a file must be executed prior to the first READ, SEEK, START, or WRITE statement for that file. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file. The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record.

When checking or writing the first label, the OPEN statement causes the user's beginning label subroutine to be executed if one is specified by a USE sentence in the Declaratives.

For an indexed output file, execution of the OPEN statement causes the user's error declarative to be executed (if one was specified) when the space allocated for the cylinder index or master index is not large enough to contain the number of entries required for the prime data.

The REVERSED and the NO REWIND options can be used only with sequential single reel files. The REVERSED option may be specified only for a file containing fixed-length (F mode) records.

Files with nonstandard header labels must not be opened for reversed reading unless the last header label is followed by a tape mark.

For tape files, the following rules apply:

1. When neither the REVERSED nor the NO REWIND option is specified, execution of the OPEN statement causes the file to be positioned at its beginning.
2. When either the REVERSED or the NO REWIND option is specified, execution of the OPEN statement does not cause the file to be repositioned. When the REVERSED option is specified, the file must have been previously positioned at its end. When the NO REWIND option is specified, the file must have been previously positioned at its beginning.

When the REVERSED option is specified, subsequent READ statements for the file make the data records of the file available in reversed order; that is, starting with the last record.

If an input file is designated with the OPTIONAL clause in the File Control paragraph of the Environment Division (sequential file processing), the clause is treated as comments. The desired effect is achieved by specifying the IGN parameter in the ASSGN control statement for the file. If the file is not present, the first READ statement for this file causes control to be passed to the imperative statement in the AT END phrase.

The I-O option permits the opening of a mass storage file for both input and output operations. Since this option implies the existence of

START Statement

the file, it cannot be used if the mass storage file is being initially created.

When the I-O option is used, the execution of the OPEN statement includes the following steps:

1. The label is checked.
2. The user's label subroutine, if one is specified by a USE sentence, is executed.
3. The label is written.

A file may be opened as INPUT and OUTPUT and I-O in any order (with intervening CLOSE statements without the UNIT or REEL option). However, an indexed file may not be opened OUTPUT and INPUT or I-O within the same program.

START Statement

The START statement initiates processing of a segment of a sequentially accessed indexed file at a specified key.

Format 1
<u>START</u> file-name <u>INVALID</u> KEY imperative-statement

Format 2 (Version 3)
<u>START</u> file-name <u>KEY</u> IS $\left\{ \begin{array}{l} \text{EQUAL TO} \\ = \end{array} \right\}$ identifier
<u>INVALID</u> KEY imperative-statement

Normally, an indexed file in the sequential mode is processed sequentially from the first record to the last or until the file is closed. If processing is to begin at other than the first record, a START statement must be executed after the OPEN but before the first READ statement. Processing will then continue sequentially until a START statement or a CLOSE statement is executed or until the end-of-file is reached.

If processing is to begin at the first record, a START statement is not required before the first READ.

File-name: The file-name must be defined by a file description entry in the Data Division.

Format 1: When Format 1 is used, the contents of the NOMINAL KEY are used as the key value of the record at which processing is to begin. In this instance, this key value must be placed in the data-name specified by the NOMINAL KEY clause for this file before the START statement is issued.

When the INVALID KEY option is specified, control is passed to the imperative-statement following INVALID KEY when the contents of the NOMINAL KEY field are invalid. The key is considered invalid when the record is not found in the file.

Program Product Information (Version 3)

Format 2: When Format 2 is used, the programmer requests that processing begin with the first record of a specified generic key class.

Identifier contains the generic key value for the request, and may be any data item less than or equal in length to the RECORD KEY for the file. Identifier may not appear in the record description for this file.

When the KEY option is specified, then before a START statement is issued, the user must place the desired value (the generic key) into identifier. When the START statement is executed, the contents of identifier are used to position the file to the beginning of the generic group. This is defined as the first record in the file with a key equal to identifier, the comparison starting with the leftmost character of the RECORD KEY, and controlled by the length of identifier. Sequential processing of the file resumes at the first record whose RECORD KEY contains a match with the contents of identifier. If the generic group does not exist, the file is positioned to the beginning of the next higher generic group in the collating sequence.

Identifiers of different lengths may be specified for different START statements for the same file.

For example, if the data records in a file contain a 10-character RECORD KEY field, and the user wishes to process the file from the beginning of a generic class defined by the first five characters within the RECORD KEY field, then he specifies a 5-character identifier field. If he later wishes to begin processing from the beginning of another generic class defined by the first three characters within the RECORD KEY field, his next START statement may specify a 3-character identifier field.

Note that upon execution of a Format 2 START statement the contents of the NOMINAL KEY field associated with the file are changed.

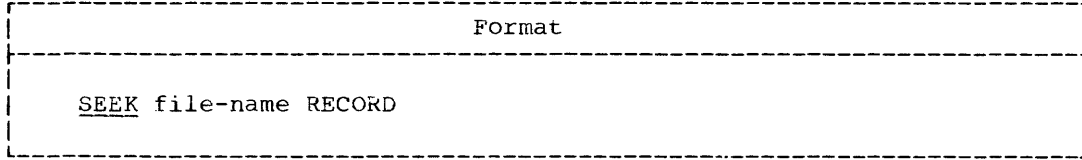
If identifier is greater in length than the NOMINAL KEY field, then the excess low-order characters of identifier are truncated.

In Format 2, when the INVALID KEY option is specified, control is passed to the imperative-statement following INVALID KEY when the contents of identifier are invalid. Identifier is considered invalid when the generic key class it contains is greater than the highest key in the file.

SEEK Statement

SEEK Statement

The SEEK statement is meant to initiate the accessing of a mass storage data record for subsequent reading or writing. It is used to optimize programming efficiency.



A SEEK statement pertains only to direct files in the random access mode and may be executed prior to the execution of a READ or WRITE statement.

The file-name must be defined by a file description entry in the Data Division.

The SEEK statement uses the contents of the data-name in the ACTUAL KEY clause for the location of the record to be accessed. At the time of execution, the determination is made as to the validity of the contents of the ACTUAL KEY data item for the particular mass storage file. If the key is invalid, when the next READ or WRITE statement for the associated file is executed, control will be given to the imperative-statement in the INVALID KEY option.

Two SEEK statements for the same direct file may logically follow each other. Any validity check associated with the first SEEK statement is negated by the execution of the second SEEK statement.

If the contents of the ACTUAL KEY are altered between the SEEK statement and the subsequent READ or WRITE statement, any validity check associated with the SEEK statement is negated, and the READ or WRITE statement is processed as if no SEEK statement preceded it.

READ Statement

The functions of the READ statement are:

1. For sequential file processing, to make available the next logical record from an input file and to give control to a specified imperative-statement when end-of-file is detected.
2. For random file processing, to make available a specific record from a mass storage file and to give control to a specified imperative-statement if the contents of the associated ACTUAL KEY or NOMINAL KEY data item are found to be invalid.

Format	
<pre> READ file-name RECORD [<u>INTO</u> identifier] { <u>AT END</u> { <u>INVALID KEY</u> } } imperative-statement </pre>	

An OPEN statement must be executed for the file prior to the execution of the first READ for that file.

When a READ statement is executed, the next logical record in the named file becomes accessible in the input area defined by the associated record description entry.

The record remains in the input area until the next input/output statement for that file is executed. No reference can be made by any statement in the Procedure Division to information that is not actually present in the current record. Thus, it is not permissible to refer to the nth occurrence of data that appears fewer than n times. If such a reference is made, no assumption should be made about the results in the object program.

When a file consists of more than one type of logical record, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information that is present in the current record is accessible.

FILE-NAME: The file-name must be defined by a file description entry in the Data Division.

INTO IDENTIFIER OPTION: The INTO identifier option makes the READ statement equivalent to a READ statement plus a MOVE statement. Identifier must be the name of a Working-Storage or Linkage Section entry, or an output record of a previously opened file. When this option is used, the current record becomes available in the input area, as well as in the area specified by identifier. Data will be moved into identifier in accordance with the COBOL rules for the MOVE statement without the CORRESPONDING option.

AT END OPTION: The AT END option must be specified for all files in the sequential access mode. If, during the execution of a READ statement, the logical end of the file is reached, control is passed to the imperative-statement specified in the AT END phrase. After execution of the imperative-statement associated with the AT END phrase, a READ statement for that file must not be given without prior execution of a CLOSE statement and an OPEN statement for that file.

READ Statement

If an input file is designated with the OPTIONAL clause in the File Control paragraph of the Environment Division (sequential file processing), the clause is treated as comments. The desired effect is achieved by specifying the IGN parameter in the ASSGN control statement for the file. If the file is not present, the first READ statement for this file causes the imperative-statement in the AT END phrase to be executed.

If, during the processing of a multivolume file in the sequential access mode, end-of-volume is recognized on a READ, the following operations are carried out:

1. The standard ending volume label procedure and the user's ending volume label procedure, if specified by the USE sentence. The order of execution of these two procedures is specified by the USE sentence.
2. A volume switch.
3. The standard beginning volume label procedure and the user's beginning volume label procedure, if specified. The order of execution is again specified by the USE sentence.
4. The first data record of the new volume is made available.

INVALID KEY OPTION: If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL or NOMINAL KEY for the file must be set to the desired value before the execution of the READ statement.

Only the track specified in the ACTUAL KEY is searched for the record being read.

When APPLY EXTENDED-SEARCH is used, the entire cylinder is searched if the desired record cannot be found on the specified track.

For a randomly accessed file, the READ statement implicitly performs the functions of the SEEK statement, unless a SEEK statement for the file has been executed prior to the READ statement.

The INVALID KEY option must be specified for files in the random access mode. The imperative-statement following INVALID KEY is executed when the contents of the ACTUAL KEY or NOMINAL KEY field are invalid.

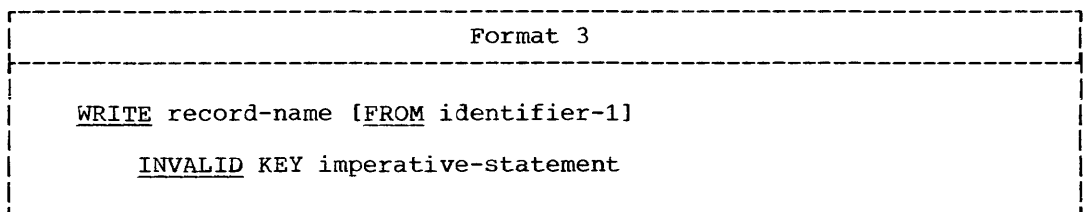
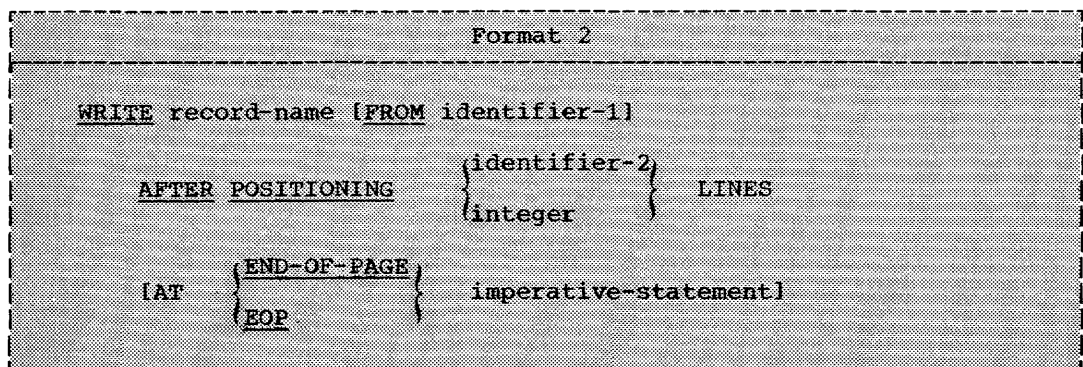
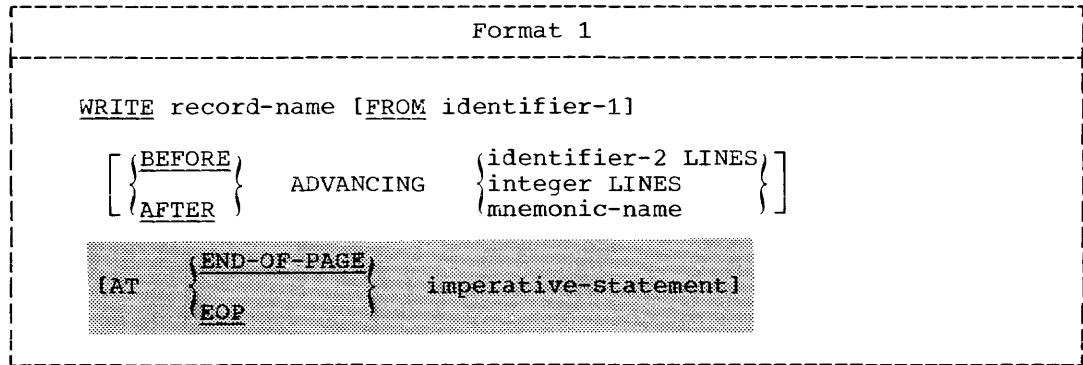
The key is considered invalid under the following conditions:

1. For a direct file that is accessed randomly, when the record is not found within the search limits (using relative addressing), or when the track address in the ACTUAL KEY field is outside the limits of the file (using actual addressing).
2. For an indexed file that is accessed randomly, when no record exists whose RECORD KEY field matches the contents of the NOMINAL KEY field.

WRITE Statement

The WRITE statement releases a logical record to an output file. It can also be used for vertical positioning of a print file. For sequentially accessed mass storage files, the WRITE statement passes control to a specified imperative-statement if the file limit is exceeded. For randomly accessed mass storage files, the WRITE statement passes control to a specified imperative-statement if the contents of the associated ACTUAL or NOMINAL KEY data item are found to be invalid.

The WRITE statement can also be used for pocket selection for a card punch file.



An OPEN statement for a file must be executed prior to executing the first WRITE statement for that file.

For files in both the sequential and random access modes, the logical record released is no longer available after the WRITE statement is executed.

RECORD-NAME: The record-name is the name of a logical record in the File Section of the Data Division and must not be part of a sort-file.

FROM OPTION: When the FROM option is written, the WRITE statement is equivalent to the statement MOVE identifier-1 TO record-name followed by the statement WRITE record-name. Data is moved into record-name in accordance with the COBOL rules for the MOVE statement without the CORRESPONDING option. Identifier-1 should be defined in the Working-Storage Section, the Linkage Section, or in another FD.

WRITE Statement

FORMAT 1 AND FORMAT 2: Formats 1 and 2 are used only with standard sequential files not residing on a mass storage device.

The ADVANCING and POSITIONING options allow control of the vertical positioning of each record on the printed page. If the ADVANCING or POSITIONING option is not used, automatic advancing is provided to cause single spacing. If the ADVANCING or POSITIONING option is used, automatic advancing is overridden.

When the ADVANCING or POSITIONING option is written for a record in a file, every WRITE statement for records in the same file must also contain one of these options. The POSITIONING and ADVANCING options may not both be specified for a file.

When the ADVANCING or POSITIONING option is used, the first character in each logical record for the file must be reserved by the user for the control character. The compiler will generate instructions to insert the appropriate carriage control character as the first character in the record. If the records are to be punched, the first character is used for pocket selection. It is the user's responsibility to see that the appropriate channels are punched on the carriage control tape.

Format 1: In the ADVANCING option, when identifier-2 is used, it must be the name of a nonnegative numeric elementary item (less than 100) described as an integer. If identifier-2 is specified, the printer page is advanced the number of lines contained in the identifier.

When integer is used in the ADVANCING option, it must be nonnegative, and less than 100. If integer is specified, the printer page is advanced the number of lines equal to the value of integer.

When the mnemonic-name option is used in the ADVANCING option, it must be defined as a function-name in the Special-Names paragraph of the Environment Division. It is used for a skip to channels 1-9, 10-12, and to suppress spacing. It is also used for pocket selection for a card punch file.

The meaning of each function-name is shown in Figure 37.

Function-name	Action Taken
CSP	Suppress spacing
C01 through C09	Skip to channel 1 through 9, respectively
C10 through C12	Skip to channel 10, 11, and 12, respectively
S01, S02	IBM 1442: pocket select 1 or 2 IBM 2540: pocket select P1 or P2
S01 through S05 (Version 3)	IBM 2560: stacker select 1 through 5 IBM 3525: stacker select 1 or 2

Figure 37. Action Taken for Function-Names -- ADVANCING Option

If the BEFORE ADVANCING option is used, the record is written before the printer page is advanced according to the preceding rules.

If the AFTER ADVANCING option is used, the record is written after the printer page is advanced according to the preceding rules.

Format 2: In the AFTER POSITIONING option, identifier-2 must be described as a one-character alphanumeric item, that is, with PICTURE X. Figure 38 shows the valid values that identifier-2 may assume and their interpretations.

In the AFTER POSITIONING option, integer must be unsigned, and must be the value 0, 1, 2, or 3. The values assume the meanings shown in Figure 39.

Value	Interpretation
b(blank)	Single-spacing
0	Double-spacing
-	Triple-spacing
+	Suppress spacing
1 - 9	Skip to channel 1 - 9, respectively
A, B, C	Skip to channel 10, 11, 12, respectively
V, W	IBM 1442: pocket select 1 (V) or 2 (W)
	IBM 2540: pocket select P1 (V) or P2 (W)
V, W, X, Y, Z (Version 3)	IBM 2560: stacker select 1 (V), 2 (W), 3 (X), 4 (Y), or 5 (Z)
	IBM 3525: stacker select 1 (V) or 2 (W)

Figure 38. Values of Identifier-2 and Their Interpretation -- POSITIONING Option

Value	Interpretation
0	Skip to channel 1 of next page (carriage control "eject")
1	Single-spacing
2	Double-spacing
3	Triple-spacing

Figure 39. Values of Integer and Their Interpretations -- POSITIONING Option

If the AFTER POSITIONING option is used, the record is written after the printer page is advanced according to the preceding rules.

END-OF-PAGE Option: The END-OF-PAGE condition exists when the channel 12 punch on the carriage control tape is sensed by an on-line printer. The printer file must be defined as an unblocked single-buffered file. The programmer should ensure that every WRITE statement in the program (whether using the ADVANCING or the POSITIONING option) advances the printer only one line at a time; otherwise, the channel 12 punch may not be sensed and results may be unpredictable.

When an END-OF-PAGE condition exists, the writing and spacing operations are completed before the END-OF-PAGE imperative-statement is executed. The END-OF-PAGE statement will be executed only for an on-line printer.

Note: DISPLAY, EXHIBIT, WRITE AFTER POSITIONING, and WRITE AFTER ADVANCING statements all cause the printer to space before printing. However, a simple WRITE statement without any option given, or a WRITE BEFORE ADVANCING statement both cause the printer to space after

WRITE Statement (Version 3)

printing. Therefore, it is possible that mixed DISPLAY statements, EXHIBIT statements, and simple WRITE statements or WRITE BEFORE ADVANCING statements within the same program may cause overprinting.

Program Product Information (Version 3)

System/370 Card Devices: The System/370 multifunction card devices have special considerations for the WRITE statement.

For punched output, the WRITE ADVANCING/POSITIONING statement can be used for stacker selection. Tables 19 and 20 show valid values.

For printed output, the END-OF-PAGE option must not be specified. In addition, any attempt to write beyond the limits of the card causes abnormal termination of the job.

For the 2560 print feature, the ADVANCING and POSITIONING options are not allowed. Automatic single spacing is provided. There may be only one WRITE statement issued for each card.

The 2560 print feature allows a maximum of 64 characters per line and up to 6 lines per card, for a maximum of 384 characters.

To position printed output on the card, the user can include SPACE characters in the output record. If, for example, he wishes to position the printing on lines 2 and 4 using the 2560 device, he can define the output record as follows:

```
01  OUTRECORD-2560.
    05  FILLER      PIC X(64)      VALUE SPACES.
    05  OUT-LINE-1  PIC X(64).
    05  FILLER      PIC X(64)      VALUE SPACES.
    05  OUT-LINE-2  PIC X(64).
    05  FILLER      PIC X(128)     VALUE SPACES.
```

The 3525 print feature allows either a 2-line print file or a multiline print file, depending on the capabilities of the specific model in use. Up to 64 characters may be printed on each line.

For a 2-line print file, the lines are printed on line 1 (top edge of card) and line 3 (between rows 11 and 12). The ADVANCING and POSITIONING options are not allowed; automatic spacing is provided. Up to two WRITE statements may be issued for each card.

For a multiline print file up to 25 lines of characters may be printed. Line control may be specified through the AFTER ADVANCING and AFTER POSITIONING options. (BEFORE ADVANCING may not be specified.)

Identifier and integer have the same meanings they have for other WRITE AFTER ADVANCING or WRITE AFTER POSITIONING statements. However, such statements must not specify space suppression, and they must not advance the line position beyond the limits of the card.

The mnemonic-name option of the WRITE AFTER ADVANCING statement may also be specified. Only the following function-names may be associated with the mnemonic-name:

WRITE Statement

<u>Function-name</u>	<u>Meaning</u>
C02	Line 3
C03	Line 5
C04	Line 7
...	...
C12	Line 23

(Note that C01 and CSP may not be used.)

(See also Appendix G: Combined Function Card Processing.)

MULTIVOLUME SEQUENTIAL FILES: The discussion below applies to all multivolume tape files and to multivolume mass storage files in the sequential access mode.

After the recognition of an end-of-volume on a multivolume OUTPUT or I-O file in the sequential access mode, the WRITE statement performs the following operations:

1. The standard ending volume label procedure and the user's ending volume label procedure if specified by a USE sentence. The order of execution of these two procedures is specified by the USE sentence.
2. A volume switch.
3. The standard beginning volume label procedures and the user's beginning volume label procedure if specified by the USE sentence. The order is specified by the USE sentence.

FORMAT 3: Format 3 is used for randomly or sequentially accessed mass storage files.

For standard sequential files opened as OUTPUT, the WRITE statement can be specified only to create the file. For such files opened as I-O, a READ statement must be executed before the WRITE statement is issued; the WRITE statement updates the record retrieved by the previous READ statement.

For sequentially accessed indexed files, the WRITE statement creates a record for an OUTPUT file.

If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL or NOMINAL KEY field for the file must be set to the desired value before the execution of a WRITE statement. For a direct file, the track specified in the ACTUAL KEY field is searched for space for the record to be written.

The INVALID KEY phrase must be specified for a file that resides on a mass storage device. Control is passed to the imperative-statement following INVALID KEY when the following conditions exist:

1. For a mass storage file in the sequential access mode opened as OUTPUT, when an attempt is made to write beyond the limit of the file. In this case, the file cannot be closed, because there is no space for an EOF record. Rerun the job with larger extents.
2. For a direct file opened as I-O or OUTPUT, if access is random and a record is being added to the file, when the track address specified in the ACTUAL KEY field is outside the limits of the file.
3. For a direct file opened as I-O, if access is random and a record is being updated, control is passed to the imperative-statement following INVALID KEY when the record is not found, or when the track number in the ACTUAL KEY field is outside the limits of the file.

REWRITE Statement

4. For an indexed file opened as OUTPUT, if access is sequential, when either one of the following conditions occurs:
 - a. The contents of the RECORD KEY field are not in ascending order when compared with the contents of the RECORD KEY field of the preceding record
 - b. The contents of the RECORD KEY field duplicate those of the preceding record
5. For an indexed file opened as I-O, if access is random and a record is being added to the file, when the contents of the NOMINAL KEY field associated with the record to be added duplicate the contents of a RECORD KEY field already in the file.

RANDOMLY ACCESSED DIRECT FILES: The WRITE statement performs the function of a SEEK statement, unless a SEEK statement for this record was executed prior to the WRITE statement.

For a randomly accessed direct file that is opened I-O, the following considerations apply:

- If A or D is specified in the ASSIGN clause system-name, then:
 - (1) a WRITE statement updates a record if the preceding input/output statement was a READ statement for a record with the same ACTUAL KEY.
 - (2) a WRITE statement adds a new record to the file, whether or not a duplicate record exists, if the preceding READ statement was not for a record with the same ACTUAL KEY.
- If U or W is specified in the ASSIGN clause system-name, then:
 - (1) a REWRITE statement searches for a record with a matching ACTUAL KEY, and updates it.
 - (2) a WRITE statement adds a new record to the file, whether or not a duplicate key exists.

REWRITE Statement

The function of the REWRITE statement is to replace a logical record on a mass storage device with a specified record, if the contents of the associated ACTUAL KEY or NOMINAL KEY are found to be valid.

Format

REWRITE record-name [FROM identifier]

INVALID KEY imperative-statement

The READ statement for a file must be executed before a REWRITE statement for the file can be executed. A REWRITE statement can be executed only for direct or indexed files opened as I-O.

If ACCESS IS RANDOM is specified for the file, the ACTUAL or NOMINAL KEY must be set to the desired value prior to the execution of the REWRITE statement.

The record-name is the name of a logical record in the File Section of the Data Division, and must not be part of a sort file.

When the FROM option is used, the REWRITE statement is equivalent to the statement MOVE identifier TO record-name followed by the statement REWRITE record-name. Identifier should be defined in the Working-Storage Section, Linkage Section, or in another FD.

For a direct file that is accessed randomly, control is passed to the imperative-statement following INVALID KEY when the contents of the ACTUAL KEY field are invalid. The key is considered invalid when the record is not found, or when the track address is outside the limits of the file.

For an indexed file that is accessed randomly, control is passed to the imperative-statement following INVALID KEY when the preceding READ statement caused an error condition.

Note: For the relationship between the REWRITE statement and the ASSIGN clause system-name, see the paragraphs on Randomly Accessed Direct files in "WRITE Statement".

ACCEPT Statement

The function of the ACCEPT statement is to obtain data from the system logical input device (SYSIPT), or from the CONSOLE.

Format							
<u>ACCEPT</u> identifier [FROM	<table border="0"> <tr> <td style="text-align: center;"> <table border="0"> <tr> <td style="text-align: center;">SYSIPT</td> <td rowspan="3" style="font-size: 2em; vertical-align: middle;">}</td> </tr> <tr> <td style="text-align: center;">CONSOLE</td> </tr> <tr> <td style="text-align: center;">mnemonic-name</td> </tr> </table> </td> <td style="text-align: center;">]</td> </tr> </table>	<table border="0"> <tr> <td style="text-align: center;">SYSIPT</td> <td rowspan="3" style="font-size: 2em; vertical-align: middle;">}</td> </tr> <tr> <td style="text-align: center;">CONSOLE</td> </tr> <tr> <td style="text-align: center;">mnemonic-name</td> </tr> </table>	SYSIPT	}	CONSOLE	mnemonic-name]
<table border="0"> <tr> <td style="text-align: center;">SYSIPT</td> <td rowspan="3" style="font-size: 2em; vertical-align: middle;">}</td> </tr> <tr> <td style="text-align: center;">CONSOLE</td> </tr> <tr> <td style="text-align: center;">mnemonic-name</td> </tr> </table>	SYSIPT	}	CONSOLE		mnemonic-name]	
SYSIPT	}						
CONSOLE							
mnemonic-name							

Identifier may be either a fixed-length group item or an elementary alphabetic, alphanumeric, external decimal, or external floating-point item. Identifier may not be any Special Register except TALLY. The data is read and the appropriate number of characters is moved into the area reserved for identifier. No editing or error checking of the incoming data is done.

If the input/output device specified by an ACCEPT statement is the same one designated for a READ statement, the results may be unpredictable.

Mnemonic-name may assume either the meaning SYSIPT or CONSOLE. Mnemonic-name must be specified in the Special-Names paragraph of the Environment Division. If mnemonic-name is associated with CONSOLE, identifier must not exceed 255 character positions in length. If the FROM option is not specified, SYSIPT is assumed.

When an ACCEPT statement with the FROM mnemonic-name for CONSOLE option or FROM CONSOLE is executed, the following actions are taken:

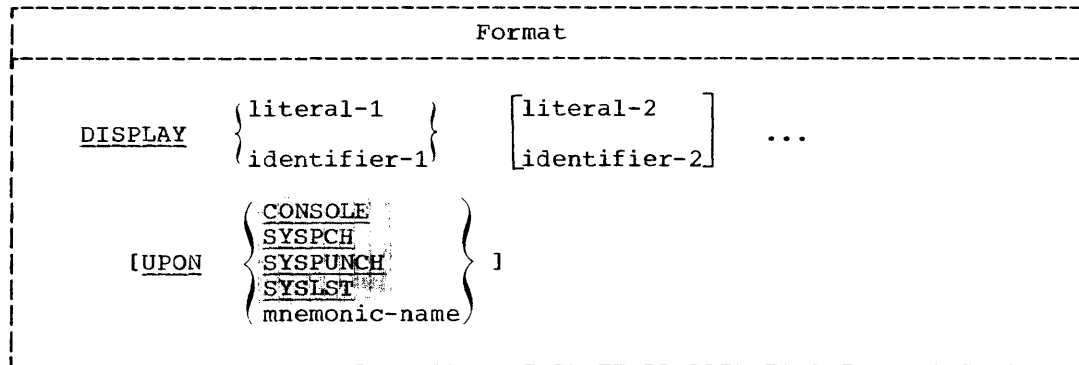
DISPLAY Statement

1. A system generated message code is automatically displayed followed by the literal "AWAITING REPLY".
2. Execution is suspended. When a console input message is identified by the control program, execution of the ACCEPT statement is resumed and the message is moved to the specified identifier and left justified regardless of the PICTURE. If the field is not filled the low-order positions may contain invalid data.

If mnemonic-name is associated with SYSIPT or the FROM SYSIPT option is specified, an input record size of 80 is assumed. If the size of the accepting data item is less than 80 characters, the data must appear as the first set of characters within the input record; any characters beyond the length of the accepting identifier are truncated. If the size of the accepting data item is greater than 80 characters, as many input records as necessary are read until the storage area allocated to the data item is filled. If the accepting data item is greater than 80 characters, but is not an exact multiple of 80, the remainder of the last input record is not accessible.

DISPLAY Statement

The function of the DISPLAY statement is to write data on an output device.



Mnemonic-name must be specified in the Special-Names paragraph of the Environment Division. Mnemonic-name may be associated only with the reserved words CONSOLE, SYSPCH, SYSPUNCH, or SYSLST.

When the UPON option is omitted, the system list device (SYSLST) is assumed.

A maximum logical record size is assumed for each device. For CONSOLE (the system logical console device), the maximum is 100 characters. For SYSLST (the system logical output device), the maximum is 120 characters. For SYSPCH or SYSPUNCH (the system punch device), the maximum is 72 characters, with positions 73-80 used for the PROGRAM-ID name.

If the total character count of all operands is less than the maximum (or 72 for SYSPCH or SYSPUNCH), the remaining character positions are padded with blanks. If the count exceeds the maximum size, operands are continued in the next record. As many records as necessary are written to display all the operands specified. Those operands pending at the time of the break are split between lines if necessary.

Identifiers described as USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or COMPUTATIONAL-3 are converted automatically to external format, as follows:

1. Internal-decimal and binary items are converted to external decimal. Only negative signed values cause a low-order sign overpunch to be developed.
2. Internal floating-point items are converted to external floating-point.
3. No other data items require conversion.

For example, if three internal-decimal items have values of -34, +34, and 34, they are displayed as 3M, 34, and 34, respectively.

If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

Identifier may not be any Special Register except TALLY.

When a DISPLAY statement contains more than one operand, the data contained in the first operand is stored as the first set of characters, and so on, until the output record is filled. This operation continues until all information is displayed. Data contained in an operand may extend into subsequent records.

Note: DISPLAY, EXHIBIT, WRITE AFTER POSITIONING, and WRITE AFTER ADVANCING statements all cause the printer to space before printing. However, a simple WRITE statement without any option given, or a WRITE BEFORE ADVANCING statement both cause the printer to space after printing. Therefore, it is possible that mixed DISPLAY statements, EXHIBIT statements, and simple WRITE statements or WRITE BEFORE ADVANCING statements within the same program may cause overprinting.

CLOSE Statement

The CLOSE statement terminates the processing of input and output reels, units, and files, with optional rewind and/or lock where applicable.

Format					
<u>CLOSE</u>	file-name-1	[REEL UNIT]	[WITH	{ NO REWIND LOCK }]
	[file-name-2	[REEL UNIT]	[WITH	{ NO REWIND LOCK }]] ...

Each file-name is the name of a file upon which the CLOSE statement is to operate; it must not be the name of a sort-file. The file-name must be defined by a file-description entry in the Data Division.

CLOSE Statement

The REEL and WITH NO REWIND options are applicable only to tape files. The UNIT option is applicable only to mass storage files in sequential access mode.

A file may be closed more than once, but each CLOSE statement (without the REEL/UNIT option) must be preceded by an OPEN statement for that file. A file which is opened in a run unit must be closed within that run unit.

For purposes of showing the effect of various CLOSE options as applied to various storage media, all input/output files are divided into the following categories:

1. Unit record volume. A file whose input or output medium is such that rewinding, units, and reels have no meaning.
2. Sequential single-volume. A sequential file that is entirely contained on one volume. There may be more than one file on this volume.
3. Sequential multivolume. A sequential file that may be contained on more than one volume.
4. Random single-volume. A file in the random access mode that may be contained on a single mass storage volume.
5. Random multivolume. A file in the random access mode that may be contained on more than one mass storage volume.

Note: See also "File Processing Summary" in the Environment Division, and "Appendix D: Summary of File Processing Techniques and Applicable Statements and Clauses".

Sequential File Processing

The results of executing each CLOSE option for each type of file are summarized in Figure 40. The definitions of the symbols in the illustration are given below. Where the definition of the symbol depends on whether the file is an input or output file, alternate definitions are given; otherwise, a definition applies to INPUT, OUTPUT, and I-O files.

A--Previous Volumes Unaffected

All volumes in the file prior to the current volume are processed according to standard volume switch procedures except those volumes controlled by a prior CLOSE REEL/UNIT statement.

B--No Rewind of Current Reel

The current volume is left in its current position.

C--Standard Close File

Files Opened as INPUT and I-O: If the file is positioned at its end and there is an ending label record, the standard ending label procedures and the user's ending label procedures (if specified by a USE sentence) are performed. System closing procedures are then performed.

If the file is positioned at its end and there is no ending label record, system closing procedures are performed.

If the file is positioned other than at its end, system closing procedures are performed but there is no ending label processing. An INPUT or an I-O file is considered to be at end-of-file if the imperative-statement in the AT END phrase has been executed and no CLOSE statement has been executed.

Files Opened as OUTPUT: If an ending label record has been described for the file, it is constructed and written on the output medium. System closing procedures are performed.

D--Standard Reel/Unit Lock

This feature has no meaning in this system and is treated as comments.

E--Standard File Lock

The compiler ensures that this file cannot be opened again during this execution of the object program. Magnetic tapes are unloaded.

F--Standard Close Volume

Files Opened as INPUT and I-O: The following operations are performed:

1. A volume switch.
2. The standard beginning volume label procedure and the user's beginning volume label procedure (if specified by the USE sentence). The order of execution of these two procedures is specified by the USE sentence.
3. Makes the next data record on the new volume available to be read.

Files Opened as OUTPUT: The following operations are performed:

1. The standard ending volume label procedure and the user's ending volume label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.
2. A volume switch.
3. The standard beginning volume label procedure and the user's beginning volume label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

G--Rewind

The current volume is positioned at its beginning.

X--Illegal

This is an illegal combination of a CLOSE option and a file type. The results at object time may be unpredictable.

CLOSE Statement

CLOSE Option \ FILE Type	Unit Record	Sequential Single-Volume	Sequential Multivolume
CLOSE	C	C, G	C, G, A
CLOSE WITH LOCK	C, E	C, G, E	C, G, E, A
CLOSE WITH NO REWIND	X	C, B	C, B, A
CLOSE REEL	X	X	F, G
CLOSE REEL WITH LOCK	X	X	F, D, G
CLOSE REEL WITH NO REWIND	X	X	F, B
CLOSE UNIT	X	X	F
CLOSE UNIT WITH LOCK	X	X	F, D

Figure 40. Relationship of Types of Sequential Files and the Options of the CLOSE Statement

General Considerations: A file is designated as optional by specifying the IGN parameter in the ASSGN control statement. If an optional file is not present, the standard end-of-file processing is not performed. For purposes of language consistency, the OPTIONAL phrase of the SELECT clause should be specified for this type of file.

If a CLOSE statement without the REEL or UNIT option has been executed for a file, the next input/output statement to be executed for that file must be an OPEN statement.

Random File Processing

The results of executing each CLOSE option for each type of file are summarized in Figure 41. The definitions of the symbols in the figure are given below. Where the definition depends on whether the file is an INPUT or OUTPUT file, alternate definitions are given; otherwise, a definition applies to INPUT, OUTPUT and I-O files.

H--Standard Close File

Files Opened as INPUT and I-O: If there is an ending label record, the ending label record is checked, and the conventional system closing procedures are performed. If there is no ending label record, the system closing procedures are performed. For I-O files, the label is updated and written.

Files Opened as OUTPUT: If an ending label record has been described for the file, it is constructed and written on the output medium. The system closing procedures are performed.

J--Standard File Lock

The compiler ensures that this file cannot be opened again during this execution of the object program.

CLOSE Option	FILE Type	Random Single-Volume	Random Multivolume
CLOSE		H	H
CLOSE WITH LOCK		H, J	H, J

Figure 41. Relationship of Types of Random Files and the Options of the CLOSE Statement

CALL Statement

SUBPROGRAM LINKAGE STATEMENTS

Subprogram linkage statements are special statements that permit communication between object programs. These statements are CALL, ENTRY, GOBACK, and EXIT.

CALL Statement

The CALL statement permits communication between a COBOL object program and one or more COBOL subprograms or other language subprograms.

Format
<code>CALL literal [USING identifier-1 [identifier-2] ...]</code>

Literal is a nonnumeric literal and is the name of the program that is being called, or the name of an entry point in the called program. The program in which the CALL statement appears is the calling program. Literal must conform to the rules for formation of a program-name. The first eight characters of literal are used to make the correspondence between the called and calling program.

When the called program is to be entered at the beginning of the Procedure Division, literal must specify the program-name in the PROGRAM-ID paragraph of the called program, and the called program must have a USING clause as part of its Procedure Division header if there is a USING clause in the CALL statement that invoked it.

When the called program is to be entered at entry points other than the beginning of the Procedure Division, these alternate entry points are identified by an ENTRY statement and a USING option corresponding to the USING option of the invoking CALL statement. In the case of a CALL statement with a corresponding ENTRY, literal must be a name other than the program-name but follows the same rules as those for the formation of a program-name.

The identifiers specified in the USING option of the CALL statement indicate those data items available to a calling program that may be referred to in the called program.

When the called subprogram is a COBOL program, each of the operands in the USING option of the calling program must be defined as a data item in the File Section, Working-Storage Section, or Linkage Section. If the called subprogram is written in a language other than COBOL, the operands of the USING option of the calling program may additionally be a file-name or a procedure-name. If the operand of the USING option is a file-name, the file with which the file-name is associated must be opened in the calling program.

Names in the two USING lists (that of the CALL in the main program and that of the Procedure Division header or the ENTRY in the subprogram) are paired in a one-for-one correspondence. In the case of index-names, no such correspondence is established.

ENTRY Statement

There is no necessary relationship between the actual names used for such paired names, but the data descriptions must be equivalent. When a group data item is named in the USING list of a Procedure Division header or an ENTRY statement, names subordinate to it in the subprogram's Linkage Section may be employed in subsequent subprogram procedural statements.

When group items with level numbers other than 01 are specified, proper word-boundary alignment is required if subordinate items are described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2.

The USING option should be included in the CALL statement only if there is a USING option in the called entry point, which is either included in the Procedure Division header of the called program or included in an ENTRY statement in the called program. The number of operands in the USING option of the CALL statement should be the same as the number of operands in the USING option of the Procedure Division header, or ENTRY statement. If the number of operands in the USING option of the CALL statement is greater than the number in the USING option in the called program, only those specified in the USING option of the called program may be referred to by the called program.

The execution of a CALL statement causes control to pass to the called program. The first time a called program is entered, its state is that of a fresh copy of the program. Each subsequent time a called program is entered, the state is as it was upon the last exit from that program. Thus, the reinitialization of the following items is the responsibility of the programmer:

- GO TO statements which have been altered
- TALLY
- Data items
- ON statements
- PERFORM statements
- EXHIBIT CHANGED statements
- EXHIBIT CHANGED NAMED statements

EXHIBIT CHANGED and EXHIBIT CHANGED NAMED operands will be compared against the value of the item at the time of its last execution, whether or not that execution was during another CALL to this program. If a branch is made out of the range of a PERFORM, after which an exit is made from the program, the range of that PERFORM is still in effect upon a subsequent entry.

Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

A called program may not be segmented.

ENTRY Statement

The ENTRY statement establishes an entry point in a COBOL subprogram.

Format
<u>ENTRY</u> literal [<u>USING</u> identifier-1 [identifier-2] ...]

USING Option

Control is transferred to the ENTRY point by a CALL statement in an invoking program.

Literal must not be the name of the called program, but is formed according to the same rules followed for program-names. Literal must not be the name of any other entry point or program-name in the run unit.

A called program, once invoked, is entered at that ENTRY statement whose operand, literal, is the same as the literal specified in the CALL statement that invoked it.

USING Option

The USING option makes data items defined in a calling program available to a called program. The number of operands in the USING option of a called program must be less than or equal to the number of operands in the corresponding CALL statement of the invoking program.

The USING option may be specified in the CALL statement, the ENTRY statement, or in the Procedure Division header.

Format 1 (Within a Calling Program)

```
CALL literal [USING identifier-1 [identifier-2]...]
```

Format 2 (Within a Called Program)

Option 1

```
ENTRY literal [USING identifier-1 [identifier-2]...]
```

Option 2

```
PROCEDURE DIVISION [USING identifier-1 [identifier-2]...].
```

When the USING option is specified in the CALL statement, it must appear on either the Procedure Division header of the called program, or in an ENTRY statement in the called program.

The USING option may be present on the Procedure Division header or in an ENTRY statement, if the object program is to function under the control of a CALL statement, and the CALL statement contains a USING option.

When a called program has a USING option on its Procedure Division header and linkage was effected by a CALL statement where literal is the name of the called program, execution of the called program begins with the first instruction in the Procedure Division after the Declaratives Section.

When linkage to a called program is effected by a CALL statement where literal is the name of an entry point specified in the ENTRY statement of the called program, that execution of the called program begins with the first statement following the ENTRY statement.

Each of the operands in the USING option of the Procedure Division header or the ENTRY statement must have been defined as a data item in the Linkage Section of the program in which this header or ENTRY statement occurs, and must have a level number of 01 or 77. Since the compiler assumes that each level-01 item is aligned upon a double-word boundary, it is the programmer's responsibility to ensure proper alignment.

When the USING option is present, the object program operates as though each occurrence of identifier-1, identifier-2, etc., in the Procedure Division had been replaced by the corresponding identifier from the USING option in the CALL statement of the calling program. That is, corresponding identifiers refer to a single set of data which is available to the calling program. The correspondence is positional and not by name. In the case of index-names, no such correspondence is established.

The following is an example of a calling program with the USING option:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. CALLPROG.
.
.

DATA DIVISION.
.
.

WORKING-STORAGE SECTION.
01 RECORD-1.
   03 SALARY          PICTURE S9(5)V99.
   03 RATE            PICTURE S9V99.
   03 HOURS           PICTURE S99V9.
.
.

PROCEDURE DIVISION.
.
.
CALL "SUBPROG" USING RECORD-1.
.
.
CALL "PAYMASTR" USING RECORD-1.
.
.

```

USING Option

The following is an example of a called subprogram associated with the preceding calling program:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. SUBPROG.  
.  
.  
.  
DATA DIVISION.  
.  
.  
.  
LINKAGE SECTION.  
01 PAYREC.  
   02 PAY                PICTURE S9(5)V99.  
   02 HOURLY-RATE        PICTURE S9V99.  
   02 HOURS              PICTURE S99V9.  
.  
.  
.  
PROCEDURE DIVISION USING PAYREC.  
.  
.  
.  
   GOBACK.  
   ENTRY "PAYMASTR" USING PAYREC.  
.  
.  
.  
   GOBACK.
```

Processing begins in CALLPROG, which is the calling program. When the statement

```
CALL "SUBPROG" USING RECORD-1.
```

is executed, control is transferred to the first statement of the Procedure Division in SUBPROG, which is the called program. In the calling program, the operand of the USING option is identified as RECORD-1.

When SUBPROG receives control, the values within RECORD-1 are made available to SUBPROG; in SUBPROG, however, they are referred to as PAYREC. Note that the PICTURE clauses for the subfields of PAYREC (described in the Linkage Section of SUBPROG) are the same as those for RECORD-1.

When processing within SUBPROG reaches the first GOBACK statement, control is returned to CALLPROG at the statement immediately following the original CALL statement. Processing then continues in CALLPROG until the statement

```
CALL "PAYMASTR" USING RECORD-1.
```

is reached. Control is again transferred to SUBPROG, but this time processing begins at the statement following the ENTRY statement in SUBPROG. The values within RECORD-1 are again made available to SUBPROG through the matching USING operand PAYREC. When processing reaches the second GOBACK statement, control is returned to CALLPROG at the statement immediately following the second CALL statement.

In any given execution of these two programs, if the values within RECORD-1 are changed between the time of the first CALL and the second, the values passed at the time of the second CALL statement will be the changed, not the original, values. If the programmer wishes to use the original values, then he must ensure that they have been saved.

Program Termination Considerations

There are three ways in COBOL source language to terminate a program. They are:

1. EXIT PROGRAM
2. GOBACK
3. STOP RUN

Figure 42 shows the effect of each program termination statement based on whether it is issued within a main program or a subprogram.

A main program is the highest level COBOL program invoked in a step. A subprogram is a COBOL program that is invoked by another COBOL program. (Programs written in other languages that follow COBOL linkage conventions are considered COBOL programs in this sense.)

Termination Statement	Main Program	Subprogram
EXIT PROGRAM	Non-operational	Return to invoking program
STOP RUN	Return to system and cause end of job step (EOJ macro)	Return to system and cause end of job step (EOJ macro)
GOBACK	Abnormal termination of job	Return to invoking program

Figure 42. Effect of Program Termination Statements Within Main Programs and Subprograms

EXIT PROGRAM Statement

This form of the EXIT statement marks the logical end of a called program.

```
Format  
paragraph-name. EXIT PROGRAM.
```

The EXIT statement must be preceded by a paragraph-name and be the only statement in the paragraph.

If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

If control reaches an EXIT PROGRAM statement and no CALL statement is active, control passes through the exit point to the first sentence of the next paragraph.

GOBACK Statement

The GOBACK statement marks the logical end of a called program.

```
Format  
GOBACK.
```

A GOBACK statement must appear as the only statement or as the last of a series of imperative-statements in a sentence.

If control reaches a GOBACK statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

If control reaches a GOBACK statement and no CALL statement is active, there will be an abnormal termination of the job.

STOP RUN Statement

For a discussion of the STOP statement with the RUN option, see "Procedure Branching Statements."

COMPILER-DIRECTING STATEMENTS

Compiler directing statements are special statements that provide instructions for the COBOL compiler. The compiler directing statements are COPY, ENTER, and NOTE.

COPY Statement

Prewritten source program entries can be included in a COBOL program at compile time. Thus, an installation can utilize standard file descriptions, record descriptions, or procedures without having to repeat programming them. These entries and procedures are contained in user-created libraries. They are included in a source program by means of a COPY statement (see "Source Program Library Facility").

ENTER Statement

The ENTER statement serves only as documentation and is intended to provide a means of allowing the use of more than one source language in the same source program. This compiler allows no other source language in the program.

Format
<u>ENTER</u> language-name [routine-name].

The ENTER statement is accepted as comments.

NOTE Statement

The NOTE statement allows the programmer to write commentary which will be produced on the source listing, but not compiled.

Format
<u>NOTE</u> character string

Any combination of the characters from the EBCDIC set may be included in the character string.

NOTE Statement

If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be part of the character string. Proper format rules for paragraph structure must be observed.

If a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the first instance of a period followed by a space.

Explanatory comments may be inserted on any line within a source program by placing an asterisk in column 7 of the line. Any combination of the characters from the EBCDIC set may be included in Area A and Area B of that line. The asterisk and the characters will be produced on the listing, but serve no other purpose.

- SORT FEATURE

- REPORT WRITER FEATURE

- TABLE HANDLING FEATURE

- SEGMENTATION FEATURE

- SOURCE PROGRAM LIBRARY FACILITY

- DEBUGGING LANGUAGE
- FORMAT CONTROL OF THE SOURCE PROGRAM LISTING
- STERLING CURRENCY FEATURE



SORT FEATURE

The COBOL programmer can gain convenient access to the sorting capability of the system sort/merge program by including a SORT statement and other elements of the Sort Feature in his source program. The Sort Feature provides the capability for sorting files and including procedures for special handling of these files both before and after they have been sorted. Within the limits of object-time storage, a source program may have any number of SORT statements, and each SORT statement may have its own special procedures.

The basic elements of the Sort Feature are the SORT statement in the Procedure Division and the Sort-File-Description (SD) entry, with its associated record description entries, in the Data Division. A sorting operation is based on sort-keys named in the SORT statement. A sort-key specifies the field within a record on which the file is sorted. Sort-keys are defined in the record description associated with the SD entry. The records of a file may be sorted in ascending or descending order or in a mixture of the two; that is, the sort-keys may be specified as ascending or descending, independent of one another, and the sequence of the sorted records will conform to the mixture specified. Additional information on the Sort Feature can be found in the Programmer's Guide.

Note: Language considerations for an ASCII-collated sort are given in Appendix E.

ELEMENTS OF THE SORT FEATURE

To use the Sort Feature, the COBOL programmer must provide additional information in the Environment, Data, and Procedure Divisions of the source program.

The SORT statement in the Procedure Division is the primary element of a source program that performs one or more sorting operations. The term "sorting operation" means not only the manipulation by the Sort Program of sort-work-files on the basis of the sort-keys designated by the COBOL programmer, but also includes the method of making records available to, and retrieving records from, these sort-work-files. A sort-work-file is the collection of records that is involved in the sorting operation as it exists on an intermediate device(s). Records are made available either by the USING or INPUT PROCEDURE options of the SORT statement. Sorted records are retrieved either by the GIVING or OUTPUT PROCEDURE options of the SORT statement.

In the Environment Division, the programmer must write SELECT sentences for all files used as input and output to the Sort Program and for the sort-file. ~~If checkpoint records are to be taken during the sorting operation, he must also include a REWIND statement.~~

In the Data Division, the programmer must include File Description entries (FD) for all files that are used to provide input to or output from the sort program and for the sort file. He must also write a Sort-File-Description (SD) entry and its associated record description entries to describe the records that are to be sorted, including their sort-key fields.

SELECT Sentence -- GIVING Option

In the Procedure Division SORT statement, the programmer specifies the sort-file to be sorted, and the sort-key names. He must also specify whether the sort is to be in ascending or descending sequence by key, and whether records are to have special processing. If there is to be such processing, he also includes in the Procedure Division the program sections that perform the processing. Special SORT registers, if used, are referenced in the Procedure Division.

ENVIRONMENT DIVISION CONSIDERATIONS FOR SORT

There are certain statements the programmer must include in the Environment Division to use the Sort Feature. Detailed descriptions of these statements follow.

INPUT-OUTPUT SECTION

The Input-Output Section is composed of two parts: the File-Control Paragraph and the I-O-Control Paragraph.

File-Control Paragraph

The File-Control Paragraph is specified once in a COBOL program. Within this paragraph, all files referred to in the source program must be named in a SELECT clause.

Files used within input and output procedures and files named in the USING and GIVING options of the SORT statement are named in the SELECT clause as described in "Environment Division." The file named in the GIVING option of the SORT statement can alternately be described in the following format.

```

                                     Format
-----
SELECT file-name
    ASSIGN TO [integer-1] system-name-1 [system-name-2] ...
        OR system-name-3 [FOR MULTIPLE { REEL } ]
                                     { UNIT }
        [RESERVE { integer-2 } ALTERNATE { AREA } 1.
                                     { NO } { AREAS }

```

The OR option is neither required nor used by this compiler, and is treated as comments.

SELECT sort-file-name

The MULTIPLE clause function is specified by object time control cards; hence, the MULTIPLE clause is neither required nor used by this compiler. The RESERVE clause is applicable as described in the "Environment Division" chapter.

Assignment of Sort Work Units

The File Control paragraph must be specified for the sort file and is used to assign work units for the sorting operation.

Format
<pre> SELECT sort-file-name ASSIGN TO [integer] system-name-1 [system-name-2] ... </pre>

sort-file-name: the name used as the first operand of the SORT statement (also the name associated with the SD entry for the sorting operation).

integer: specifies the number of work units available to the Sort Program. From one through eight units may be assigned for a disk sort. From three through nine units may be assigned for a tape sort. If integer is not specified, the compiler assumes one unit for a disk sort and three units for a tape sort.

system-name-1: created in the same format as the system-name in other ASSIGN clauses (see "Environment Division"). However, the names by which the work files are known to the Sort Program are fixed. System-name-1 must always contain the symbolic unit number SYS001. If the sort work files have standard labels, system-name-1 must specify a name field of SORTWK1. The class and organization fields are treated as comments. Class will always default to UT, and organization will always default to S.

Integer work units beginning with the first are reserved for the sorting operation. The user may, if he wishes, specify these additional work units in multiple system-names. However, the compiler treats these as documentation. Instead, the second work unit is assigned to SYS002 with name SORTWK2, the third to SYS003 with name SORTWK3, etc.

For example, the SELECT sentence for a sort-file with standard labels, which has five work units (tape) available, would be:

```

SELECT SORTFILE
      ASSIGN TO 5 SYS001-UT-2400-S-SORTWK1.
          
```

SYS001 through SYS005 are assigned by the compiler to the work units.

I-O-CONTROL Paragraph

The I-O-Control paragraph specifies when checkpoints are to be taken, as well as what core storage area is to be shared by different files.

RERUN/SAME AREA Clauses

The I-O-Control paragraph is coded once in the source program. The checkpoint interval associated with the standard RERUN format (specified in the "Environment Division") is determined by the number of records processed for the given file. However, the format has meaning only when a file is not being used in a sorting operation. Obtaining checkpoint records within the operation of a SORT statement is specified by a special format of the RERUN statement, as described below.

RERUN Clause

The format of the RERUN clause used in conjunction with the sorting operation is:

Format
<u>RERUN ON</u> system-name

The presence of this format of the RERUN clause indicates that checkpoint records are to be written, at logical intervals determined by the sort program, during the execution of all SORT statements that appear in a source program. Its absence indicates that, within the execution of any SORT statement, checkpoint records are not to be taken.

System-name must not be the same as any system-name used in a FILE-CONTROL ASSIGN clause, but must follow the same rules of formation. This system-name may be the same as the system name specified in a standard RERUN clause.

At the time the checkpoint procedure of the SORT statement takes effect, the status of all open files, whether involved in the sorting operation or not, is recorded.

SAME RECORD/SORT AREA Clause

The SAME RECORD/SORT AREA clause specifies that two or more files are to use the same storage area during processing.

Format
<u>SAME</u> { <u>RECORD</u> } AREA FOR file-name-1 {file-name-2} ... <u>SORT</u> }

The RECORD option is used when only FD or mixed FD and SD files are named. The named files share only the processing area of the current logical record. Although several of the files may be open at the same time, only the logical record of one of these files can exist in the record area at one time.

The SORT option is used when only SD files are named. Its function is to optimize the assignment of storage areas to a given SORT statement. The system will handle storage allocation automatically; hence, the SORT storage option, if given, will be ignored.

DATA DIVISION CONSIDERATIONS FOR SORT

In the Data Division the programmer must include File Description entries for files that are input to or output from the Sort, sort-file-description entries which describe the records as they appear on the sort work files, and record description entries for each.

FILE SECTION

The File Section of a program which contains a sorting operation must furnish information concerning the physical structure, identification, and record names of the sort work file. This is provided in the sort-file-description entry.

Sort-File Description

A sort-file-description entry must appear in the File Section for every file named as the first operand of a SORT statement.

Format	
<u>SD</u> sort-file-name	
[RECORDING MODE IS mode]	
[<u>DATA</u> { RECORD IS RECORDS ARE}]	data-name-1 [data-name-2] ...]
[RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]	
[<u>LABEL</u> { RECORD IS RECORDS ARE}]	{ STANDARD OMITTED } 1.

There must be a SELECT sentence for sort-file-name that describes the system-name for the sort work file. Sort-file-name is also the name specified in the SORT statement.

The RECORDING MODE clause is discussed in "Data Division." The recording mode must be F or V.

Sort -- Procedure Division Considerations

The DATA RECORDS clause specifies the names of the records in the file to be sorted. Data-name-1, data-name-2, ... of the DATA RECORDS clause refer to the records described in the record descriptions associated with this SD.

The RECORD CONTAINS clause specifies the size of data records in the file to be sorted. This clause is optional. The actual size and mode (fixed or variable) of the records to be sorted are determined from the level-01 descriptions associated with a given SD entry. When the USING and GIVING options of the SORT statement are used, the record length associated with the SD must be the same length as the record associated with the FD's for the USING and GIVING files. If any of the SD data record descriptions contains an OCCURS clause with the DEPENDING ON option, variable-length records are assumed. Refer to "Data Division" for the format assumptions that are made by the compiler when the RECORDING MODE clause is not specified.

Both the DATA RECORDS and the RECORD CONTAINS clauses are described in "Data Division."

The LABEL RECORDS clause specifies whether the sort-work-files have standard labels or no labels. If this clause is not specified and the file is assigned to a mass storage device, LABEL RECORDS STANDARD is assumed; otherwise LABEL RECORDS OMITTED is assumed.

PROCEDURE DIVISION CONSIDERATIONS FOR SORT

The Procedure Division must contain a SORT statement to describe the sorting operation and, optionally, input and output procedures. The procedure-names constituting the input and output procedures are specified within the SORT statement.

The Procedure Division may contain more than one SORT statement appearing anywhere except in the declaratives portion or in the input and output procedures associated with a SORT statement.

Note: If DISPLAY or EXHIBIT is used in an input or output procedure of a SORT verb, each time DISPLAY or EXHIBIT is executed SYSLST will be opened since it is closed by SORT. To achieve a good performance of a COBOL program, the DISPLAY and EXHIBIT verbs should only be used for debugging purposes in input or output procedures for SORT.

SORT Statement

The SORT statement provides information that controls the sorting operation. This information directs the sorting operation to obtain records to be sorted either from an input procedure or the USING file to sort the records on a set of specified sort keys, and in the final phase of the sorting operation to make each record available in sorted order, either to an output procedure or to the GIVING file.

```

Format
-----
SORT file-name-1 ON {DESCENDING} KEY {data-name-1} ...
                   {ASCENDING}
[ON {DESCENDING} KEY {data-name-2} ...] ...
   {ASCENDING}
{INPUT PROCEDURE IS section-name-1 [THRU section-name-2]}
{USING file-name-2}
{OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4]}
{GIVING file-name-3}

```

File-name-1 is the name given in the sort-file-description entry that describes the records to be sorted.

ASCENDING and DESCENDING: The ASCENDING and DESCENDING options specify whether records are to be sorted into an ascending or descending sequence, respectively, based on one or more sort keys.

Each data-name represents a "key" data item and must be described in the records associated with the sort-file-name.

At least one ASCENDING or DESCENDING clause must be specified. Both options may be specified in the same SORT statement, in which case, records are sorted on data-name-1, in ascending or descending order, and then within data-name-1, they are sorted on the KEY data item represented by data-name-2, in ascending or descending order, etc.

Keys are always listed from left to right in order of decreasing significance, regardless of whether they are ascending or descending.

The direction of the sort depends on the use of the ASCENDING or DESCENDING clauses as follows:

1. When an ASCENDING clause is used, the sorted sequence is from the lowest value of the key to the highest value, according to the collating sequence for the COBOL character set.
2. When a DESCENDING clause is used, the sorted sequence is from the highest value of the key to the lowest value, according to the collating sequence of the COBOL character set.

Sort keys must be one of the types of data item listed in Figure 43. Corresponding to each type of data item is a collating sequence that is used with it for sorting.

A character in the EBCDIC collating sequence (used with alphabetic, alphanumeric, etc., data items) is interpreted as not being signed. For fixed-point and internal floating-point numeric data items, characters are collated algebraically (that is, as being signed).

SORT Statement

Type of Data Item Used for Sort Key	Collating Sequence
Alphabetic	EBCDIC
Alphanumeric	EBCDIC
Numeric Edited	EBCDIC
Group	EBCDIC
External Decimal	Zoned Decimal
Binary	Fixed Point
Internal Decimal	Fixed point
Internal Floating point	Floating Point
External Floating point	EBCDIC

Figure 43. SORT Collating Sequences Used for Sort Keys

The EBCDIC collating sequence for COBOL characters in ascending order is:

1. (space)
2. . (period or decimal point)
3. < (less than)
4. ((left parenthesis)
5. + (plus symbol)
6. \$ (currency symbol)
7. * (asterisk)
8.) (right parenthesis)
9. ; (semicolon)
10. - (hyphen or minus symbol)
11. / (stroke, virgule, slash)
12. , (comma)
13. > (greater than)
14. ' (apostrophe or single quotation mark)
15. = (equal sign)
16. " (quotation mark)
- 17-42. A through Z
- 43-52. 0 through 9

(The complete EBCDIC collating sequence is given in IBM System/360 Reference Data, Order No. X20-1703.)

The record description for every record that is listed in the DATA RECORDS clause of an SD description must contain the "key" items data-name-1, data-name-2, etc. These "key" items are subject to the following rules:

1. Keys must be physically located in the same position and have the same data format in every logical record of the sort-file. If there are multiple record descriptions in an SD, it is sufficient to describe a key in only one of the record descriptions.
2. Key items must not contain an OCCURS clause nor be subordinate to entries that contain an OCCURS clause.
3. A maximum of 12 keys may be specified. The total length of all the keys must not exceed 256 bytes.
4. All keys must be at a fixed displacement from the beginning of a record; that is, they cannot be located following a variable table in a record.
5. All key fields must be located within the first 4092 bytes of a logical record.
6. The data-names describing the keys may be qualified.

SECTION-NAME-1 AND SECTION-NAME-2: Section-name-1 is the name of an input procedure. Section-name-2 is the name of the last section that contains the input procedure in the COBOL main program. Section-name-2 is required if the procedure terminates in a section other than that in which it was started.

INPUT PROCEDURE: The presence of the INPUT PROCEDURE option indicates that the programmer has written an input procedure to process records before they are sorted and has included the procedure in the Procedure Division as one or more distinct sections.

The input procedure must consist of one or more sections that are written consecutively and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file.

Control must not be passed to the input procedure unless a related SORT statement is being executed, because the RELEASE statement in the input procedure has no meaning unless it is controlled by a SORT statement. The input procedure can include any procedures needed to select, create, or modify records. There are three restrictions on the procedural statements within an input procedure:

1. The input procedure must not contain any SORT statements.
2. The input procedure must not contain any transfers of control to points outside the input procedure. The execution of a CALL statement to another program that follows standard linkage conventions, or the execution of USE declaratives for label handling and error processing are not considered transfers of control outside of an input procedure. Hence, they are allowed to be activated within these procedures.

However, this compiler permits the ALTER, GO TO, and PERFORM statements in the input procedure to refer to procedure-names outside the input procedure. It is the user's responsibility to insure a return to the input procedure after exiting through a GO TO or PERFORM statement.

3. The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure (with the exception of the return of control from a declarative section).

However, this compiler allows ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division to refer to procedure-names within the Input Procedure. If a SORT statement is active when the transfer of control is made, then all such transfers are valid. If a SORT statement is not active, however, then the user must ensure that such a transfer of control does not cause:

- a. a RELEASE statement to be executed
- b. control to reach the end of the Input Procedure

If an input procedure is specified, control is passed to the input procedure when the SORT program input phase is ready to receive the first record. The compiler inserts a return mechanism at the end of the last section of the input procedure and, when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

The RELEASE statement transfers records from the Input Procedure to the input phase of the sort operation (see "RELEASE Statement").

SORT Statement

USING: If the USING option is specified, all the records in file-name-2 are transferred automatically to file-name-1. At the time of execution of the SORT statement, file-name-2 must not be open. File-name-2 must be a standard sequential file.

For the USING option, the compiler will open, read, release, and close file-name-2 without the programmer specifying these functions. If the user specifies error handling and/or label processing declaratives for file-name-2, the compiler will make the necessary linkage to the appropriate declarative section.

SECTION-NAME-3 AND SECTION-NAME-4: Section-name-3 represents the name of an output procedure. Section-name-4 is the name of the last section that contains the output procedure in the COBOL main program. Section-name-4 is required if the procedure terminates in a section other than that in which it is started.

OUTPUT PROCEDURE: The output procedure must consist of one or more sections that are written consecutively and do not form a part of any input procedure. The output procedure must include at least one RETURN statement in order to make sorted records available for processing.

Control must not be passed to the output procedure unless a related SORT statement is being executed, because RETURN statements in the output procedure have no meaning unless they are controlled by a SORT statement. The output procedure may consist of any procedures needed to select, modify, or copy the records that are being returned one at a time, in sorted order, from the sort-file. There are three restrictions on the procedural statements within the output procedure.

1. The output procedure must not contain any SORT statements.
2. The output procedure must not contain any transfers of control to points outside the output procedure. The execution of a CALL statement to another program that follows standard linkage conventions, or the execution of USE declaratives for label handling and error processing are not considered transfers of control outside of an output procedure. Hence, they are allowed to be activated within these procedures.

However, this compiler permits the ALTER, GO TO, and PERFORM statements in the output procedure to refer to procedure-names outside the output procedure. It is the user's responsibility to insure a return to the output procedure after exiting through a GO TO or PERFORM statement.

3. The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedure (with the exception of the return of control from a declarative section).

However, this compiler allows ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division to refer to procedure-names within the Output Procedure. If a SORT statement is active when the transfer of control is made, then all such transfers are valid. If a SORT statement is not active, however, then the user must ensure that such a transfer of control does not cause:

- a. a RETURN statement to be executed
- b. control to reach the end of the Output Procedure

If an output procedure is specified, control passes to it after file-name-1 has been placed in sequence by the SORT statement. The

compiler inserts a return mechanism at the end of the last section in the output procedure. When control passes the last statement in the output procedure, the return mechanism provides for termination of the SORT and then passes control to the next statement after the SORT statement.

When all the records are sorted, control is passed to the output procedure. The RETURN statement in the output procedure is a request for the next record (see "RETURN Statement").

GIVING: If the GIVING option is used, all sorted records in file-name-1 are automatically transferred to file-name-3. At the time of execution of the SORT statement, file-name-3 must not be open. file-name-3 must name a standard sequential file.

For the GIVING option, the compiler will open, return, write, and close file-name-3 without the programmer specifying these functions. If the user specifies error handling and/or label processing declaratives for file-name-3, the compiler will make the necessary linkage to the appropriate declarative section.

CONTROL OF INPUT OR OUTPUT PROCEDURES: The INPUT or OUTPUT PROCEDURE options function in a manner similar to the PERFORM statement (Option 1); for example, naming a section in an INPUT PROCEDURE clause causes execution of that section during the sorting operation to proceed as though that section had been the subject of a PERFORM statement. As in the execution of a PERFORM statement, the execution of the section is terminated after execution of its last statement. The procedure may be terminated by an EXIT statement (see "EXIT Statement").

RELEASE Statement

The RELEASE statement transfers records from the Input Procedure to the input phase of the Sort operation.

Format
<u>RELEASE</u> sort-record-name [<u>FROM</u> identifier]

A RELEASE statement may be used only within the range of an input procedure associated with a SORT statement.

If the INPUT PROCEDURE option is specified, the RELEASE statement must be included within the given set of procedures.

Sort-record-name must be the name of a logical record in the associated sort-file description.

When the FROM identifier option is used, it makes the RELEASE statement equivalent to the statement MOVE identifier TO sort-record-name, followed by the statement RELEASE.

RETURN/EXIT Statements

Sort-record-name and identifier must not refer to the same storage area. A move with the rules for group items is effected from identifier, using the length of the record-name associated with the SD entry.

After the RELEASE statement is executed, the logical record is no longer available. When control passes from the input procedure, the file consists of those records that were placed in it by the execution of RELEASE statements.

RETURN Statement

The RETURN statement obtains individual records in sorted order from the final phase of the sort program.

```
Format
-----
RETURN sort-file-name RECORD [INTO identifier]
      AT END imperative-statement
```

Sort-file-name is the name given in the sort-file-description entry that describes the records to be sorted.

All references to records retrieved by a RETURN statement must be in terms of the record description(s) associated with the SD entry, unless the INTO option is specified. The retrieved record may, optionally, be moved to the user's own area and be referenced as appropriate.

A RETURN statement may only be used within the range of an output procedure associated with a SORT statement for file-name-1.

The identifier must be the name of a working-storage area or an output record area. Use of the INTO option has the same effect as the MOVE statement for alphanumeric items.

The imperative-statement in the AT END phrase specifies the action to be taken when all the sorted records have been obtained from the sorting operation.

After execution of the imperative-statement in the AT END phrase, no RETURN statements may be executed within the current output procedure.

EXIT Statement

The EXIT statement may be used as a common end point for input or output procedures as with procedures executed through a PERFORM statement.

```
Format
-----
paragraph-name. EXIT.
```


When used in this manner, the EXIT statement must appear as the only statement in the last paragraph of the input or output procedure.

SPECIAL REGISTERS FOR SORT

Four special registers are available to users of the Sort Feature. These registers provide a means of object time communication between the user and the Sort Feature, and they aid in the optimization of sort performance.

The first three registers may have control information transferred to them at object time if the user specifies them as the receiving fields of statements such as MOVE. However, none of the Sort registers can be used as operands in ACCEPT, DISPLAY, or EXHIBIT statements. The information must be passed before the SORT statement is executed. The registers are initialized to zero by the compiler, but are not reset after a sort procedure is executed. Thus, if a program has multiple SORT statements, any values in the registers at the completion of one sorting operation will be in the registers at the beginning of another, unless modified.

1. SORT-FILE-SIZE is the name of a binary data item whose PICTURE is S9(8). It is used for the estimated number of records in the file to be sorted. If SORT-FILE-SIZE is omitted, the Sort Feature assumes that the file contains the maximum number of records that can be processed with the available core size and number of work units. If the estimate exceeds the maximum, the estimate will be ignored.
2. SORT-CORE-SIZE is the name of a binary data item whose PICTURE is S9(8). It is used to specify the number of bytes of storage available to the sorting operation if it is different from the core size that the Sort Feature would normally use.
3. SORT-MODE-SIZE is the name of a binary data item whose PICTURE is S9(5). It is used for variable-length records. If the length of most records in the file is significantly different from the average record length, performance is improved by specifying the most frequently occurring record length. If SORT-MODE-SIZE is omitted, the average length is assumed. For example, if records vary in length from 20 to 100 bytes, but most records are 30 bytes long, the value 30 should be moved to SORT-MODE-SIZE. The maximum record length handled by the Sort Feature is 32,000 bytes.
4. SORT-RETURN is the name of a binary data item whose PICTURE is S9(4). It contains a return code of 0 or 16 at the completion of a sorting operation to signify the success or failure, respectively, of the sort operation.

An example of how variable information can be passed to the Sort Feature by use of a register is:

```
ACCEPT FILE-SIZE FROM CONSOLE.  
MOVE FILE-SIZE TO SORT-FILE-SIZE.
```

SAMPLE PROGRAM USING THE SORT FEATURE

This example (Figure 44) illustrates a sort based on a sales contest. The records to be sorted contain data on salesmen: name and address, employee number, department number, and pre-calculated net sales for the contest period.

The salesman with the highest net sales in each department wins a prize, and smaller prizes are awarded for second highest sales, third highest, etc. The order of the SORT is (1) by department, the lowest numbered first (ASCENDING KEY DEPT); and (2) by net sales within each department, the highest net sales first (DESCENDING KEY NET-SALES).

The records for the employees of departments 7 and 9 are eliminated in an input procedure (SCREEN-DEPT) before sorting begins. The remaining records are then sorted, and the output is placed on another file for use in a later job step.

```

|000005 IDENTIFICATION DIVISION.
|000010 PROGRAM-ID. CONTEST.
|000015 ENVIRONMENT DIVISION.
|000016 CONFIGURATION SECTION.
|000017 SOURCE-COMPUTER. IBM-360-F50.
|000018 OBJECT-COMPUTER. IBM-360-F50.
|000019 SPECIAL-NAMES. SYSLST IS PRINTER.
|000020 INPUT-OUTPUT SECTION.
|000025 FILE-CONTROL.
|000030     SELECT NET-FILE-IN ASSIGN TO SYS008-UT-2400-S.
|000035     SELECT NET-FILE-OUT ASSIGN TO SYS007-UT-2400-S-SORTOUT.
|000040     SELECT NET-FILE ASSIGN TO 3 SYS001-UT-2400-S.
|000050 DATA DIVISION.
|000055 FILE SECTION.
|000060 SD NET-FILE
|000065     DATA RECORD IS SALES-RECORD.
|000070 01 SALES-RECORD.
|000075     02 EMPL-NO           PICTURE 9(6).
|000080     02 DEPT              PICTURE 9(2).
|000085     02 NET-SALES        PICTURE 9(7)V99.
|000090     02 NAME-ADDR        PICTURE X(55).
|000095 FD NET-FILE-IN
|000096     LABEL RECORDS ARE OMITTED
|000100     DATA RECORD IS NET-CARD-IN.
|000105 01 NET-CARD-IN.
|000110     02 EMPL-NO-IN        PICTURE 9(6).
|000115     02 DEPT-IN          PICTURE 9(2).
|000120     02 NET-SALES-IN      PICTURE 9(7)V99.
|000125     02 NAME-ADDR-IN     PICTURE X(55).
|000130 FD NET-FILE-OUT
|000131     LABEL RECORDS ARE OMITTED
|000135     DATA RECORD IS NET-CARD-OUT.
|000140 01 NET-CARD-OUT.
|000145     02 EMPL-NO-OUT      PICTURE 9(6).
|000150     02 DEPT-OUT          PICTURE 9(2).
|000155     02 NET-SALES-OUT    PICTURE 9(7)V99.
|000160     02 NAME-ADDR-OUT    PICTURE X(55).

```

Figure 44. Sample Program Using the SORT Feature (Part 1 of 2)

```

|000165 PROCEDURE DIVISION.
|000170 ELIM-DEPT-7-9-NO-PRINTOUT.
|000175     SORT NET-FILE
|000180         ASCENDING KEY DEPT,
|000185         DESCENDING KEY NET-SALES
|000190         INPUT PROCEDURE SCREEN-DEPT
|000195         GIVING NET-FILE-OUT.
|000200 CHECK-RESULTS SECTION.
|000205 C-R-1.
|000210     OPEN INPUT NET-FILE-OUT.
|000215 C-R-2.
|000220     READ NET-FILE-OUT AT END GO TO C-R-FINAL.
|000225     DISPLAY EMPL-NO-OUT DEPT-OUT NET-SALES-OUT
|000230         NAME-ADDR-OUT UPON PRINTER.
|000235 C-R-3.
|000240     GO TO C-R-2.
|000245 C-R-FINAL.
|000250     CLOSE NET-FILE-OUT.
|000255     STOP RUN.
|000260 SCREEN-DEPT SECTION.
|000265 S-D-1.
|000270     OPEN INPUT NET-FILE-IN.
|000275 S-D-2.
|000280     READ NET-FILE-IN AT END GO TO S-D-FINAL.
|000285     DISPLAY EMPL-NO-IN DEPT-IN NET-SALES-IN
|000290         NAME-ADDR-IN UPON PRINTER.
|000295 S-D-3.
|000300     IF DEPT-IN = 7 OR 9 GO TO S-D-2
|000305         ELSE
|000310             MOVE NET-CARD-IN TO SALES-RECORD,
|000315             RELEASE SALES-RECORD,
|000320             GO TO S-D-2.
|000325 S-D-FINAL.
|000330     CLOSE NET-FILE-IN.
|000335 S-D-END.
|000340     EXIT.

```

Figure 44. Sample Program Using the SORT Feature (Part 2 of 2)

REPORT WRITER FEATURE

The Report Writer Feature permits the programmer to specify the format of a printed report in the Data Division, thereby minimizing the amount of Procedure Division coding he would have to write to create the report.

A printed report consists of the information reported and the format in which it is printed. Several reports can be produced by one program.

In the Data Division, the programmer gives the name(s) and describes the format(s) of the report(s) he wishes produced. In the Procedure Division, he writes the statements that produce the report(s).

At program execution time, the report in the format defined is produced -- data to be accumulated is summed, totals are produced, counters are stepped and reset, and each line and each page is printed. Thus, the programmer need not concern himself with the details of these operations.

DATA DIVISION -- OVERALL DESCRIPTION

In the Data Division, the programmer must write an FD entry that names the output file upon which the report is to be written, and must also name the report itself. A report may be written on two files at the same time.

At the end of the Data Division, he must add a Report Section to define the format of each report named. In the Report Section, there are two types of entries:

1. The Report Description Entry (RD) which describes the physical aspects of the report format.
2. The report group description entries which describe the data items within the report and their relation to the report format.

In the report description entry, the programmer specifies the maximum number of lines per page, where report groups are to appear on the page, and which data items are to be considered as controls.

Controls govern the basic format of the report. When a control changes value -- that is, when a control break occurs -- special actions will be taken before the next line of the report is printed. Controls are listed in a hierarchy, proceeding from the most inclusive down to the least inclusive. Thus, by specifying HEADING and FOOTING controls, the programmer is able to instruct the Report Writer to produce the report in whatever format he desires.

For example, in the program at the end of this chapter, the hierarchy of controls proceeds from the highest (FINAL) to an intermediate control (MONTH) to the minor control (DAY-1). DAY-1 is the minor control since, if MONTH changes, DAY-1 also must change. Whenever any control changes, special actions are performed by the Report Writer -- sum information is totaled, counters are reset, special information is printed, and so forth -- before the next detail line is printed.

The report group description entries describe the characteristics of all data items contained within the report group: the format of each data item present, its placement in relation to the other data items within the report group, and any control factors associated with the

group. Information to be presented within a report group can be described in three ways:

- as SOURCE information, which is information from outside the report
- as SUM information, which is the result of addition operations upon any data present, whether SOURCE information or other SUM information
- as VALUE information, which is constant information

Through the RD and the report group description entries, the programmer has thus defined completely the content, the format, and the summing operations necessary to produce the desired report.

PROCEDURE DIVISION -- OVERALL DESCRIPTION

In the Procedure Division, the programmer instructs the Report Writer to produce the report through the use of three Report Writer statements: INITIATE, GENERATE, and TERMINATE.

The INITIATE statement performs functions in the Report Writer analogous to the OPEN statement for individual files.

The GENERATE statement automatically produces the body of the report. Necessary headings and footings are printed, counters are incremented and reset as desired, source information is obtained, and sum information is produced, data is moved to the data item(s) in the report group description entry, controls are tested, and when a control break occurs, the additional lines requested are printed, as well as the detail line that caused the control break. All of this is done automatically, thus relieving the programmer of the responsibility for writing detailed tests and looping procedures that would otherwise be necessary.

The TERMINATE statement completes the processing of a report. It is analogous to the CLOSE statement for individual files.

In the Declaratives portion of the Procedure Division, the programmer may also specify a USE BEFORE REPORTING procedure for report group. In this procedure, he is able to specify any additional processing he wishes done before a specific report group is printed.

Two special registers are used by the Report Writer feature:

LINE-COUNTER -- which is a numeric counter used by the Report Writer to determine when a PAGE HEADING and/or a PAGE FOOTING report group is to be presented. The maximum value of LINE-COUNTER is based on the number of lines per page as specified in the PAGE LIMIT(S) clause. LINE-COUNTER may be referred to in any Procedure Division statement.

PAGE-COUNTER -- which is a numeric counter that may be used as a SOURCE data item in order to present the page number on a report line. The maximum size of PAGE-COUNTER is based on the size specified in the PICTURE clause associated with an elementary item whose SOURCE IS PAGE-COUNTER. This counter may be referred to by any Procedure Division statement.

Figure 46, at the end of this chapter, gives an example of a Report Writer program for a manufacturer's quarterly report.

FD Entry/REPORT Clause

Figure 47, which follows the program, shows the report that would be produced.

DATA DIVISION CONSIDERATIONS FOR REPORT WRITER

The names of all the reports to be produced must be named in the File Section of the Data Division. An entry is required in the FD entry to list the names of the reports to be produced on that file. A Report Section must be added at the end of the Data Division to define the format of each report.

FILE DESCRIPTION

The File Description furnishes information concerning the physical structure, identification, and record-names pertaining to a given file.

General Format
<u>FD</u> file-name
[BLOCK CONTAINS Clause]
[RECORD CONTAINS Clause]
[RECORDING MODE Clause]
LABEL RECORDS Clause
[VALUE OF Clause]
[DATA RECORDS Clause]
REPORT Clause.

A discussion of all the above-mentioned clauses appears in "Data Division." A description of the REPORT clause, the RECORDING MODE clause, the DATA RECORDS clause, and the RECORD CONTAINS clause for a file on which a report is produced follows.

REPORT Clause

Each unique report-name must appear in the REPORT clause of the FD entry (or entries) for the file(s) on which the report(s) is to be produced. The REPORT clause cross references the description of Report Description entries with their associated File Description entry.

Format	
{ <u>REPORT IS</u> }	report-name-1 [report-name-2]...
{ <u>REPORTS ARE</u> }	

RECORDING MODE/DATA RECORDS/RECORD CONTAINS Clauses

Each File Description entry for standard sequential OUTPUT files within the File Section may include a REPORT clause containing the names of one or more reports. These reports may be of different sizes, formats, etc., and the order in which their names appear in the clause is not significant.

Each unique report-name listed in an FD entry must be the subject of an RD entry in the Report Section. A given report-name may appear in a maximum of two REPORT clauses.

RECORDING MODE Clause

The RECORDING MODE clause is used to specify the format of the logical records within the file. If this clause is omitted, the recording mode is determined as described in "Data Division."

DATA RECORDS Clause

If the DATA RECORDS clause is specified, and the file is used for output, the AFTER ADVANCING option must be used when writing records named in this clause.

RECORD CONTAINS Clause

The RECORD CONTAINS clause enables the user to specify the maximum size of his report record.

Format
<u>RECORD</u> CONTAINS [<u>integer-1</u> <u>TO</u>] <u>integer-2</u> CHARACTERS

The specified size of each report record must include the carriage control/line spacing character, and the CODE character, if the CODE option is used. If the RECORD CONTAINS clause is omitted, the compiler assumes a default size of 133 characters.

For variable-length records, the size of each print line will be integer-2 characters, and the size of each blank line required for spacing will be 17 characters. For fixed-length records, the size of each print line and each blank line required for spacing will be integer-2 characters.

For further information on the RECORD CONTAINS clause, see "Data Division."

RD (Report Description) Entry

REPORT SECTION

The Report Section consists of two types of entries for each report; one describes the physical aspects of the report format, the other type describes conceptual characteristics of the items that make up the report and their relationship to the report format. These are:

1. Report Description entry (RD)
2. Report group description entries

The Report Section must begin with the header REPORT SECTION.

Report Description Entry

The Report Description entry contains information pertaining to the overall format of a report named in the File Section and is uniquely identified by the level indicator RD. The clauses that follow the name of the report are optional, and their order of appearance is not significant.

The entries in this section stipulate:

1. The maximum number of lines that can appear on a page.
2. Where report groups are to appear on a page.
3. Data items that act as control factors during presentation of the report.

General Format
<pre>REPORT SECTION. RD report-name [CODE Clause] [CONTROL Clause] [PAGE LIMIT Clause].</pre>

RD is the level indicator.

Report-name is the name of the report and must be unique. The report-name must be specified in a REPORT clause in the File Description entry for the file on which the report is to be written.

CODE Clause

The CODE clause is used to specify an identifying character added at the beginning of each line produced. The identification is meaningful when more than one report is written on a file.

Format
<u>WITH CODE</u> mnemonic-name

Mnemonic-name must be associated with a single character literal used as function-name-1 in the SPECIAL-NAMES paragraph in the Environment Division. The identifying character is appended to the beginning of the line, preceding the carriage control/line spacing character. This clause should not be specified if the report is to be printed on-line.

CONTROL Clause

The CONTROL clause indicates the identifiers that specify the control hierarchy for this report, that is, the control breaks.

Format						
<table> <tr> <td>{ <u>CONTROL IS</u> }</td> <td>{ <u>FINAL</u></td> </tr> <tr> <td>{ <u>CONTROLS ARE</u> }</td> <td>{ identifier-1 [identifier-2]... }</td> </tr> <tr> <td></td> <td>{ <u>FINAL</u> identifier-1 [identifier-2]... }</td> </tr> </table>	{ <u>CONTROL IS</u> }	{ <u>FINAL</u>	{ <u>CONTROLS ARE</u> }	{ identifier-1 [identifier-2]... }		{ <u>FINAL</u> identifier-1 [identifier-2]... }
{ <u>CONTROL IS</u> }	{ <u>FINAL</u>					
{ <u>CONTROLS ARE</u> }	{ identifier-1 [identifier-2]... }					
	{ <u>FINAL</u> identifier-1 [identifier-2]... }					

A control is a data item that is tested each time a detail report group is generated. If the test indicates that the value of the data item (i.e., CONTROL) has changed, a control break is said to occur, and special action (described below) is taken before printing the detail line.

FINAL is the highest level control. (It is the one exception to the statement that controls are data items.) The identifiers specify the control hierarchy of the other controls. Identifier-1 is the major control, identifier-2 is the intermediate control, etc. The last identifier specified is the minor control. The levels of the controls are indicated by the order in which they are written. FINAL need not be specified in the CONTROL clause, even if a CONTROL HEADING FINAL or CONTROL FOOTING FINAL is specified.

The control identifiers must each specify different data items; that is, their descriptions must not be such that they occupy (partially or completely) the same area of storage, or overlap in any way.

When controls are tested, the highest level control specified is tested first, then the second highest level, etc. When a control break is found for a particular level, a control break is implied for each lower level as well. A control break for FINAL occurs only at the beginning and ending of a report (i.e., before the first detail line is printed and after the last detail is printed).

The action to be taken as a result of a control break depends on what the programmer defines. He may define a CONTROL HEADING report group and/or a CONTROL FOOTING group or neither for each control.

PAGE LIMIT Clause

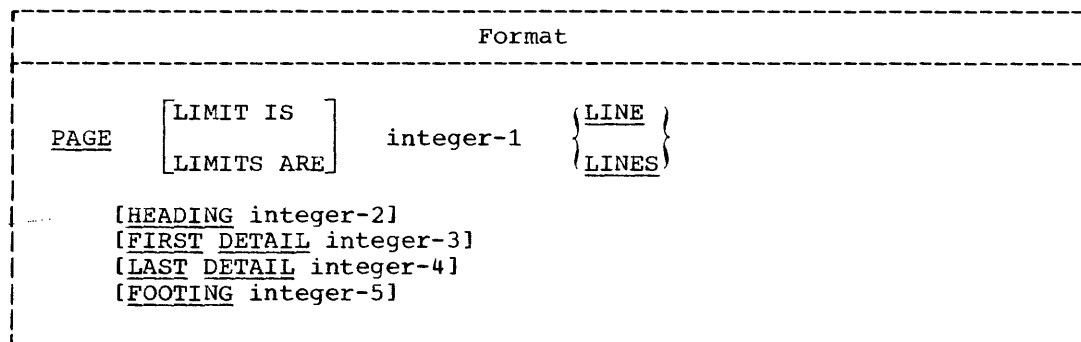
The control footings and headings that are defined are printed prior to printing the originally referenced detail. They are printed in the following order: lowest level control footing, next higher level control footing, etc., up to and including the control footing for the level at which the control break occurred; then the control heading for that level, then the next lower level control heading, etc., down to and including the minor control heading; then the detail is printed. If, in the course of printing control headings and footings, a page condition is detected, the current page is ejected and a new page begun. If the associated report groups are given, a page footing and/or a page heading are also printed.

The CONTROL clause is required when CONTROL HEADING or CONTROL FOOTING report groups ~~other than FINAL~~ are specified.

The identifiers specified in the CONTROL clause are the only identifiers referred to by the RESET and TYPE clauses in a report group description entry for this report. The identifiers must be defined in the File or Working-Storage Section of the Data Division.

PAGE LIMIT Clause

The PAGE LIMIT clause indicates the specific line control to be maintained within the logical presentation of a page, i.e., it describes the physical format of a page of the report.



If this clause is not specified, PAGE-COUNTER and LINE-COUNTER special registers are not generated.

The PAGE LIMIT clause is required when page format must be controlled by the Report Writer.

integer-1: The PAGE LIMIT integer-1 LINES clause is required to specify the depth of the report page; the depth of the report page may or may not be equal to the physical perforated continuous form often associated in a report with the page length. The size of the fixed data-name, LINE-COUNTER, is the maximum numeric size based on integer-1 lines required for the counter to prevent overflow.

integer-2: The first line number of the first heading print group is specified by integer-2. No print group will start preceding integer-2, i.e., integer-2 is the first line on which anything may be printed.

- integer-3: The first line number of the first normal print group (body group) is specified by integer-3. No DETAIL, CONTROL HEADING, or CONTROL FOOTING print group will start before integer-3.
- integer-4: The last line number of the last nonfooting body group is specified by integer-4. No DETAIL or CONTROL HEADING print group will extend beyond integer-4.
- integer-5: The last line number of the last CONTROL FOOTING print group is specified by integer-5. No CONTROL FOOTING print group will extend beyond integer-5. PAGE FOOTING print groups will follow integer-5.

Using the parameters of the PAGE LIMIT clause, the Report Writer establishes the areas of the page where each type of report group is allowed to be printed. The following are the page areas for each type of report group:

1. A REPORT HEADING report group can extend from line integer-2 to line integer-1, inclusive. If the REPORT HEADING report group is not on a page by itself, the FIRST DETAIL integer-3 clause must be present in the PAGE LIMIT clause of the report.
2. A PAGE HEADING report group may extend from line integer-2 to line integer-3 minus 1, inclusive. If a PAGE HEADING report group is specified in the report description, the FIRST DETAIL integer-3 clause must be present in the PAGE LIMIT clause of the report. A PAGE HEADING report group that follows a REPORT HEADING report group on the same page must be able to be printed in the area of the page defined in this rule.
3. CONTROL HEADING report groups and DETAIL report groups must be printed in the area of the page that extends from line integer-3 to line integer-4, inclusive.
4. CONTROL FOOTING report groups must be printed in the area of the page extending from line integer-3 to line integer-5, inclusive.
5. A PAGE FOOTING report group may extend from line integer-5 plus 1 to line integer-1, inclusive. If PAGE FOOTING is specified in the report description, either the FOOTING integer-5 or LAST DETAIL integer-4 clause must be present in the PAGE LIMIT clause of the report.
6. A REPORT FOOTING report group can extend from line integer-2 to line integer-1, inclusive. If the REPORT FOOTING report group is not on a page by itself, either the FOOTING integer-5 or LAST DETAIL integer-4 clause must be present in the PAGE LIMIT clause of the report.

Figure 45 pictorially represents page format report group control when the PAGE LIMIT clause is specified.

PAGE LIMIT Clause

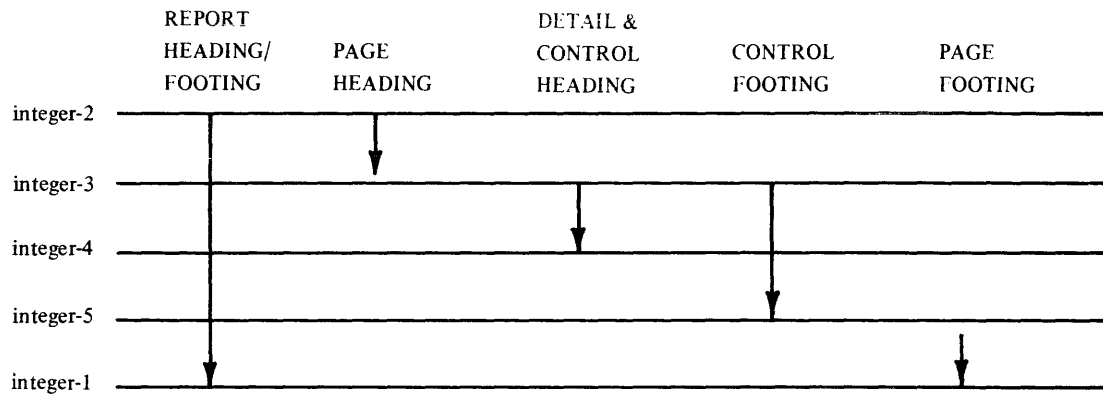


Figure 45. Page Format When the PAGE LIMIT Clause is Specified

The PAGE LIMIT clause may be omitted when no association is desired between report groups and the physical format of an output page. In this case, relative line spacing must be indicated for all report groups of the report.

If absolute line spacing is indicated for all the report groups, none of the integer-2 through integer-5 controls need be specified. If any of these limits are specified for a report that has only absolute line spacing, the limits are ignored.

If relative line spacing is indicated for any report group, all LINE NUMBER and NEXT GROUP spacing must be consistent with the controls specified or implied in the PAGE LIMIT clause.

If PAGE LIMITS integer-1 is specified and some or all of the HEADING integer-2, FIRST DETAIL integer-3, LAST DETAIL integer-4, FOOTING integer-5 clauses are omitted, the following implicit control is assumed for all omitted specifications:

1. If HEADING integer-2 is omitted, integer-2 is considered to be equivalent to the value 1, that is, LINE NUMBER one.
2. If FIRST DETAIL integer-3 is omitted, integer-3 is considered to be equivalent to the value of integer-2.
3. If LAST DETAIL integer-4 is omitted, integer-4 is considered to be equivalent to the value of integer-5.
4. If FOOTING integer-5 is omitted, integer-5 is considered to be equivalent to the value of integer-4. If both LAST DETAIL integer-4 and FOOTING integer-5 are omitted, integer-4 and integer-5 are both considered to be equivalent to the value of integer-1.

Only one PAGE LIMIT clause may be specified for a Report Description entry.

- Integer-1 through integer-5 must be positive integers.
- Integer-2 through integer-5 must be in ascending order. Integer-5 must not exceed integer-1.

Report Group Description Entry

A report comprises one or more report groups. Each report group is described by a hierarchy of entries similar to the description of a data record. There are three categories of report groups: heading groups, detail groups, and footing groups. A CONTROL HEADING, DETAIL, or CONTROL FOOTING report group may also be referred to as a body group.

The report group description entry defines the format and characteristics for a report group. The relative placement of a particular report group within the hierarchy of report groups, the format of all items, and any control factors associated with the group are defined in this entry.

Schematically, a report group is a line, a series of lines, or a null (i.e., nonprintable) group. A report group is considered to be one unit of the report. Therefore, the lines of a report group are printed as a unit.

A null group is a report group for which no LINE or COLUMN clauses have been specified (that is, a nonprintable report group).

The report group description entry defines the format and characteristics applicable to the type of report group.

1. For all report groups that are not null groups, the description entry indicates where and when the report group is to be presented.
2. For all report groups, the description entry indicates when the nonprinting functions of the report group, such as summation, are to be performed.
3. For all report groups except DETAIL, the description entry allows for the execution of a user-specified procedure prior to printing a report group. If a report group is a null group, the execution of the user procedure occurs in the same manner as though the report group were printed.
4. For CONTROL FOOTING report groups, the description entry indicates the user's summation algorithm.

Report group names are required when reference is made in the Procedure Division:

- to a DETAIL report group by a GENERATE statement.
- to a HEADING or FOOTING report group by a USE sentence.

Report group names are required when reference is made in the Report Section to a DETAIL report group by a SUM UPON clause.

Except for the data-name clause which, when present, must immediately follow the level number, the clauses may be written in any order.

Report-name is the only qualifier available for report groups and SUM counters.

Note: A report group description entry is not a true hierarchical structure like a record in the File or Working-Storage Sections. In the Report Section, an 01-level item does not define a continuous area of storage occupied by subordinate items. For this reason, the 01 data-name may not be used as a qualifier for what outwardly appear to be subordinate items. Since a CONTROL FOOTING in a given report may only roll forward SUM counters defined within the same RD, qualification of the SUM counter is implicit and never necessary. In the Procedure Division, a SUM counter data-name may be qualified by the appropriate RD report-name.

Report Group Description Entry--Formats

General Format 1

```
01  [data-name-1]
     [LINE Clause]
     [NEXT GROUP Clause]
     TYPE Clause
     [USAGE Clause].
```

General Format 2

```
level number [data-name-1]
             [LINE clause]
             [USAGE clause].
```

General Format 3

```
level number [data-name-1]
             [BLANK WHEN ZERO Clause]
             [COLUMN Clause]
             [GROUP Clause]
             [JUSTIFIED Clause]
             [LINE Clause]
             [PICTURE Clause]
             [RESET Clause]
             [ {SOURCE}
               {SUM   } Clause ]
             [ {VALUE }
               ]
             [USAGE Clause].
```

General Format 4

```
01  [data-name-1]
     [BLANK WHEN ZERO Clause]
     [COLUMN Clause]
     [GROUP Clause]
     [JUSTIFIED Clause]
     [LINE Clause]
     [NEXT GROUP Clause]
     PICTURE Clause
     [RESET Clause]
     [ {SOURCE}
       {SUM   } Clause
     [ {VALUE }
       ]
     TYPE Clause
     [USAGE Clause].
```

Report Group Description Entry

Format 1 is used to indicate a report group. A report group description must contain a report group entry (level-01) and it must be the first entry. A report group extends from this entry either to the next report group level-01 entry or to the end of the next report description. A null report group may contain only a Format 1 report group entry.

Format 2 is used to indicate a group item. A group item entry may contain a level number from 02 through 48; this entry has the following functions:

- If a report group has more than one line and one of the lines contains more than one elementary item, a group item entry may be used to indicate the LINE number of the subordinate elementary items.
- If a group item entry contains no LINE clause and there are no SUM counters subordinate to it, its only function is documentation.

Format 3 is used to indicate an elementary item. An elementary item entry may contain a level number from 02 through 49; this entry has the following functions:

- An elementary item entry may be used to describe an item that is to be presented on a printed line. In this case, a COLUMN clause, a PICTURE clause, and either a SOURCE, SUM, or VALUE clause must be present.
- An elementary item entry in a DETAIL report group may be used to indicate to the Report Writer what operands are to be summed upon presentation of the DETAIL report group.
- An elementary item entry in a CONTROL FOOTING report group may be used to define a SUM counter. (See SUM Clause.)

Format 4 is used to indicate a report group that consists of only one elementary item. If Format 4 is used to define the report group instead of Format 1, it must be the only entry in the group.

LINE Clause

The LINE clause indicates the absolute or relative line number of this entry in reference to the page or previous entry.

Format	
<u>LINE NUMBER IS</u>	{ <u>integer-1</u> <u>PLUS integer-2</u> <u>NEXT PAGE</u> }

Each line of a report must have a LINE clause associated with it. For the first line of a report group, the LINE clause must be given either at the report group level or prior to or for the first elementary item in the line. For report lines other than the first in a report group, the LINE clause must be given prior to or for the first elementary item in the line. When a LINE clause is encountered, subsequent entries following the entry with the LINE clause are implicitly presented on the same line until either another LINE clause or the end of the report group is encountered.

Integer-1 and integer-2 must be positive integers.

LINE NUMBER IS integer-1 is an absolute LINE clause. It indicates the fixed line of the page on which this line is to be printed. LINE-COUNTER is set to the value of integer-1 and is used for printing the items in this and the following entries within the report group until a different value for the LINE-COUNTER is specified.

LINE Clause

LINE NUMBER IS PLUS integer-2 is a relative LINE clause. The line is printed relative to the previous line either printed or skipped. LINE-COUNTER is incremented by the value of integer-2 and is used for printing the items in this and the following entries within the report group until a different value for the LINE-COUNTER is specified. Exceptions to this rule are discussed later.

LINE NUMBER IS NEXT PAGE indicates that this report group is to be printed on the next page, not on the current page. This LINE clause may appear only in a report group entry or may be the LINE clause of the first line of the report group.

Within any report group, absolute LINE NUMBER entries must be indicated in ascending order, and an absolute LINE NUMBER cannot be preceded by a relative LINE NUMBER. If the first line of the first body group that is to be printed on a page contains either a relative LINE clause or a LINE NUMBER IS NEXT PAGE clause, the line is printed on line FIRST DETAIL integer-3. However, if the LINE-COUNTER contains a value that is greater than or equal to FIRST DETAIL integer-3, the line is printed on line LINE-COUNTER plus 1. This value of LINE-COUNTER was set by an absolute NEXT GROUP clause in the previously printed body group (see rules for NEXT GROUP).

If the report group entry of a body group contains a LINE NUMBER IS NEXT PAGE clause and the first line contains a relative LINE clause, the first line is printed relative to either FIRST DETAIL integer-3 or LINE-COUNTER, whichever is greater. This value of LINE-COUNTER was set by an absolute NEXT GROUP clause in the previously printed body group.

The following are the rules for the LINE clause by report group type:

1. REPORT HEADING

- LINE NUMBER IS NEXT PAGE cannot be specified in the report group.
- The first line of the report group may contain an absolute or relative LINE clause.
- If the first line contains a relative line clause, it is relative to HEADING integer-2.

2. PAGE HEADING

- LINE NUMBER IS NEXT PAGE cannot be specified in the report group.
- The first line may contain either an absolute or relative LINE clause.
- If the first line contains a relative LINE clause, it is relative to either HEADING integer-2 or the value of LINE-COUNTER, whichever is greater. The value in LINE-COUNTER that is greater than HEADING integer-2 can only result from a REPORT HEADING report group being printed on the same page as the PAGE HEADING report group.

3. CONTROL HEADING, DETAIL, and CONTROL FOOTING

- LINE NUMBER IS NEXT PAGE may be specified in the report group.
- The first line of the report group may contain either an absolute or relative LINE clause.

4. PAGE FOOTING

- LINE NUMBER IS NEXT PAGE cannot be specified in the report group.
- The first line of the report group may contain an absolute or relative LINE clause.
- If the first line contains a relative LINE clause, it is relative to FOOTING integer-5.

5. REPORT FOOTING

- If the report group is to be printed on a page by itself, LINE NUMBER IS NEXT PAGE must be specified.
- If LINE NUMBER IS NEXT PAGE is the only LINE clause in the report group description, the line will be printed on line HEADING integer-2.
- If the report group description does not contain a LINE NUMBER IS NEXT PAGE clause, the first line must contain an absolute or relative LINE clause. If it contains a relative LINE clause, the line is relative to either FOOTING integer-5 or the value of LINE-COUNTER, whichever is greater. The value in LINE-COUNTER that is greater than FOOTING integer-5 can only result from the printing of the PAGE FOOTING report group.

NEXT GROUP Clause

The NEXT GROUP clause indicates the spacing condition following the last line of the report group.

Format	
<u>NEXT GROUP IS</u>	{ integer-1 PLUS integer-2 NEXT PAGE }

The NEXT GROUP clause can appear only in a report group entry. integer-1 and integer-2 must be positive integers.

NEXT GROUP Clause

The following are the rules for the NEXT GROUP clause by report group type:

1. REPORT HEADING

- If the report group is to be printed on a page by itself, NEXT GROUP IS NEXT PAGE must be specified in the report group description.
- Integer-1 indicates an absolute line number which sets the LINE-COUNTER to this value after printing the last line of the report group.
- Integer-2 indicates a relative line number which increments the LINE-COUNTER by the integer-2 value after printing the last line of the report group.
- An absolute or relative NEXT GROUP clause must not cause the LINE-COUNTER to be set to a value greater than FIRST DETAIL integer-3 minus 1.

2. PAGE HEADING, PAGE FOOTING, and REPORT FOOTING

- A NEXT GROUP clause cannot be specified in the report group.

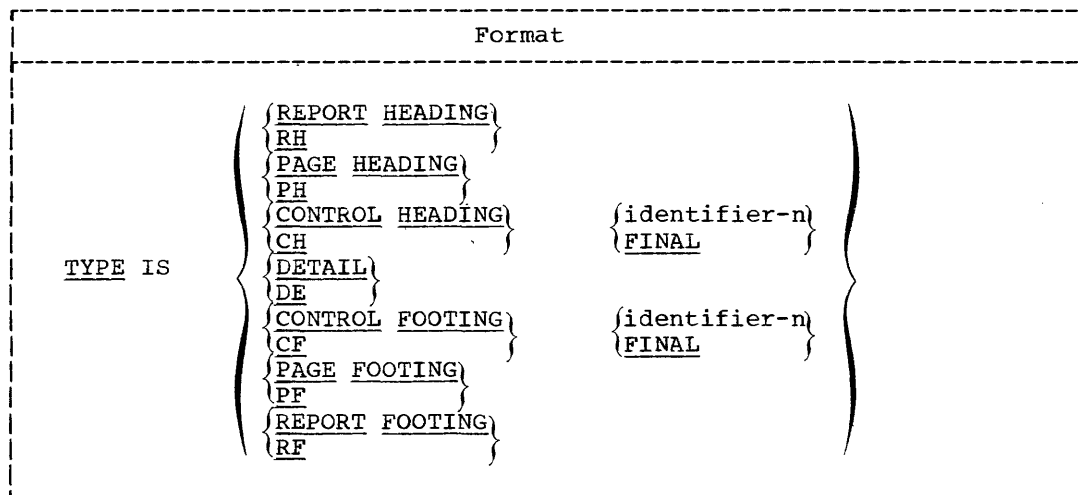
3. CONTROL HEADING, DETAIL, and CONTROL FOOTING

- If a NEXT GROUP clause implies a page change, the change occurs only when the next body group is to be printed.
- The NEXT GROUP IS NEXT PAGE clause indicates that no more body groups are to be printed on this page.
- An absolute or relative NEXT GROUP clause may cause the LINE-COUNTER to be set to a value greater than or equal to FIRST DETAIL integer-3 and less than or equal to FOOTING integer-5. This is an exception to the rule which defines the page area of CONTROL HEADING and DETAIL report groups.
- If a NEXT GROUP IS integer-1 clause causes a page change, the value of LINE-COUNTER is set to the value of integer-1 before the formatting of the first line of the next body group to be printed. This implies that if the first line of the next body group to be printed contains a relative LINE NUMBER clause, the line will be printed on line LINE-COUNTER plus 1; if the first line contains an absolute LINE NUMBER clause that is less than or equal to integer-1, a page will be printed which contains only PAGE HEADING and FOOTING report groups, and the following page will contain the body group.
- When the NEXT GROUP clause is specified for a CONTROL FOOTING report group, the NEXT GROUP clause functions are performed only when a control break occurs for the control that is associated with this report group.

If the USE procedure for a report group contains a MOVE 1 PRINT-SWITCH statement, the NEXT GROUP clause functions are not performed for this report group.

TYPE Clause

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be generated.



The TYPE clause in a particular report group entry indicates the point in time at which this report group will be generated as output.

If the report group is described as TYPE DETAIL or DE, then a GENERATE statement in the Procedure Division directs the Report Writer to produce this report group. Each DETAIL report group must have a unique data-name at level-01 in a report.

If the report group is described as other than TYPE DETAIL or DE, the generation of this report group is an automatic feature of the Report Writer, as detailed in the following paragraphs.

The REPORT HEADING or RH entry indicates a report group that is produced only once at the beginning of a report during the execution of the first GENERATE statement. There can be only one report group of this type in a report. SOURCE clauses used in REPORT HEADING report groups refer to the values of data items at the time the first GENERATE statement is executed.

The PAGE HEADING or PH entry indicates a report group that is produced at the beginning of each page according to PAGE condition rules as specified below. There can be only one report group of this type in a report.

The CONTROL HEADING or CH entry indicates a report group that is produced at the beginning of a control group for a designated identifier, or, in the case of FINAL, is produced once before the first control group during the execution of the first GENERATE statement. There can be only one report group of this type for each identifier and for the FINAL entry specified in a report. In order to produce any CONTROL HEADING report groups, a control break must occur. SOURCE clauses used in CONTROL HEADING FINAL report groups refer to the values of the items at the time the first GENERATE statement is executed.

The CONTROL FOOTING or CF entry indicates a report group that is produced at the end of a control group for a designated identifier or is

TYPE Clause

produced once at the termination of a report ending a FINAL control group. There can be only one report group of this type for each identifier and for the FINAL entry specified in a report. In order to produce any CONTROL FOOTING report groups, a control break must occur. SOURCE clauses used in CONTROL FOOTING FINAL report groups refer to the values of the items at the time the TERMINATE statement is executed.

The PAGE FOOTING or PF entry indicates a report group that is produced at the bottom of each page according to PAGE condition rules as specified below. There can be only one report group of this type in a report.

The REPORT FOOTING or RF entry indicates a report group that is produced only at the termination of a report. There can be only one report group of this type in a report. SOURCE clauses used in TYPE REPORT FOOTING report groups refer to the value of items at the time the TERMINATE statement is executed.

Identifier-n, as well as FINAL, must be one of the identifiers described in the CONTROL clause in the Report Description entry.

A FINAL type control break may be designated only once for CONTROL HEADING or CONTROL FOOTING entries within a particular report description.

Nothing precedes a REPORT HEADING entry and nothing follows a REPORT FOOTING entry within a report.

The HEADING or FOOTING report groups occur in the following Report Writer sequence if all exist for a given report:

```
REPORT HEADING (one occurrence only)
PAGE HEADING
.
.
.
CONTROL HEADING
DETAIL
CONTROL FOOTING
.
.
.
PAGE FOOTING
REPORT FOOTING (one occurrence only)
```

CONTROL HEADING report groups are presented in the following hierarchical arrangement:

```
Final Control Heading (one occurrence only)
Major Control Heading
.
.
.
Minor Control Heading
```

CONTROL FOOTING report groups are presented in the following hierarchical arrangement:

```
Minor Control Footing
.
.
.
Major Control Footing
Final Control Footing (one occurrence only)
```

CONTROL HEADING report groups appear with the current values of any indicated SOURCE data items before the DETAIL report groups of the CONTROL group are produced. CONTROL FOOTING report groups appear with the previous values of any indicated SOURCE data items specified in the CONTROL clause, just after the DETAIL report groups of that CONTROL group have been produced.

The USE procedures specified for a CONTROL FOOTING report group that refer to:

- source data items that are specified in the CONTROL clause affect the previous value of the items
- source data items that are not specified in the CONTROLS clause affect the current value of the items

These report groups appear whenever a control break occurs. LINE NUMBER determines the absolute or relative position of the CONTROL report groups exclusive of the other HEADING and FOOTING report groups.

USAGE Clause

DISPLAY is the only option that may be specified for group and elementary items in a Report Group Description entry (see "USAGE Clause").

COLUMN Clause

The COLUMN clause indicates the absolute column number on the printed page of the high-order (leftmost) character of an elementary item.

Format
<u>COLUMN NUMBER</u> IS integer

The COLUMN clause indicates that the leftmost character of the elementary item is placed in the position specified by integer. If the COLUMN clause is not specified, the elementary item, though included in the description of the report group, is suppressed when the report group is produced at object time.

Integer must be a positive integer.

The COLUMN number clause is given at the elementary level within a report group even if the elementary level is a single level-01 entry, which alone constitutes the report group.

Within a report group and a particular LINE NUMBER specification, COLUMN number entries need not be indicated from left to right.

GROUP INDICATE/RESET Clause

GROUP INDICATE Clause

The GROUP INDICATE clause specifies that this elementary item is to be produced only on the first occurrence of the item after any control or page break.

Format
<u>GROUP</u> INDICATE

The GROUP INDICATE clause must be specified only at the elementary item level within a DETAIL report group.

An elementary item is not only group indicated in the first DETAIL report group containing the item after a control break, but is also group indicated in the first DETAIL report group containing the item on a new page, even though a control break did not occur.

JUSTIFIED Clause

The JUSTIFIED clause is applicable in report group description entries as described in "Data Division."

PICTURE Clause

The PICTURE clause is applicable in Report Group Description entries as described in "Data Division."

RESET Clause

The RESET clause indicates the CONTROL identifier that causes the SUM counter in the elementary item entry to be reset to zero on a CONTROL break.

Format
<u>RESET</u> ON { identifier } <u>FINAL</u> }

After presentation of the CONTROL FOOTING report group, the counters associated with the report group are reset automatically to zero, unless

an explicit RESET clause is given specifying reset based on a higher level control than the associated control for the report group.

The RESET clause may be used for progressive totaling of identifiers where subtotals of identifiers may be desired without automatic resetting upon producing the report group.

Identifier must be one of the identifiers described in the CONTROL clause in the Report Description entry (RD). Identifier must be a higher level CONTROL identifier than the CONTROL identifier associated with the CONTROL FOOTING report group in which the SUM and RESET clauses appear.

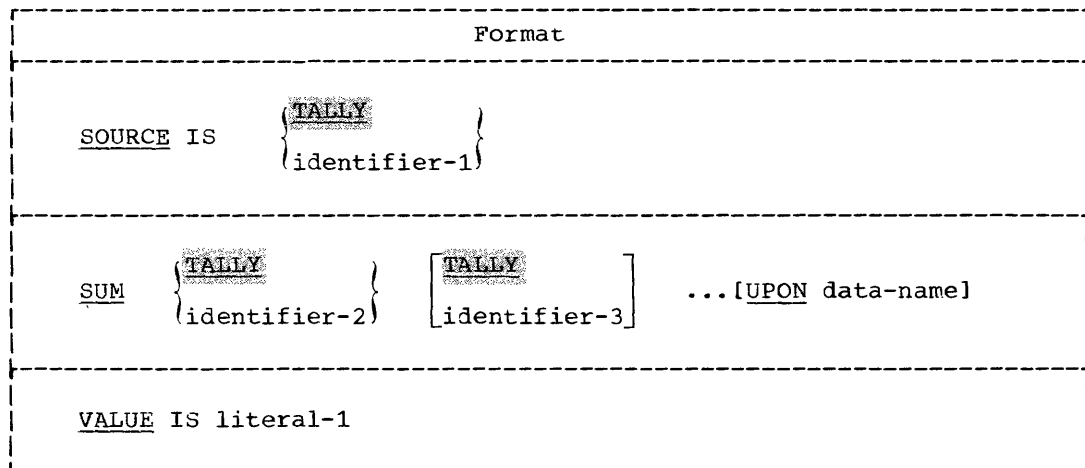
The RESET clause may be used only in conjunction with a SUM clause.

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause is applicable here as discussed in "Data Division."

SOURCE, SUM, or VALUE Clause

The SOURCE, SUM, or VALUE clause defines the purpose of this elementary item within the report group.



SOURCE: The SOURCE clause indicates a data item that is to be used as the source for this report item. The item is presented according to the PICTURE clause and the COLUMN clause in this elementary item entry.

The SOURCE clause has two functions:

1. To specify a data item that is to be printed
2. To specify a data item that is to be summed in a CONTROL FOOTING report group (see SUM clause)

TALLY may be used in place of identifier-1 in a SOURCE clause.

SOURCE/SUM/VALUE Clause

SUM: The SUM clause is used to cause automatic summation of data and may appear only in an elementary item entry of a CONTROL FOOTING report group. The presence of a SUM clause defines a SUM counter. If a SUM counter is to be referred to by a Procedure Division statement or Report Section entry, a data-name clause must be specified with the SUM clause entry. The data-name then represents the summation counter generated by the Report Writer to total the operands specified immediately following SUM. If reference is never made to a summation counter, the counter need not be named explicitly by a data-name entry.

Whether the elementary item entry that contains a SUM clause names the summation counter or not, the PICTURE clause must be specified for each SUM counter. Editing characters or editing clauses may be included in the description of a SUM counter. Editing of a SUM counter occurs only upon presentation of that SUM counter. At all other times, the SUM counter is treated as a numeric data item. The SUM counter must be large enough to accommodate the summed quantity without truncation of integral digits.

An operand of a SUM clause must be an elementary numeric data item that appears in the File, Working-Storage, or Linkage Section, or is the name of a SUM counter. Except when they are names of SUM counters, SUM operands may be qualified, subscripted or indexed. A SUM counter that is an operand of SUM clause must be defined in the same CONTROL FOOTING report group that contains this SUM clause or in a CONTROL FOOTING report group that is at a lower level in the control hierarchy of this report.

A SUM counter is incremented by its operands in the following manner:

- An operand that is an elementary numeric data item appearing in the File, Working-Storage, or Linkage Section is added to the SUM counter upon the generation of a DETAIL report group that contains this operand as a SOURCE data item; even if the operand appears in more than one SOURCE clause of the DETAIL report group, it is added only once to the SUM counter. The operands must appear exactly as they are in the SOURCE clauses with regard to qualification, subscripting, and indexing. If the SUM clause contains the UPON option, the operand does not have to appear in a SOURCE clause of the DETAIL report group.
- An operand that is a SUM counter and is defined in a CONTROL FOOTING that is at any lower level in the control hierarchy of this report is summed before presentation of the CONTROL FOOTING in which it is defined. This counter updating is commonly called rolling counters forward.
- An operand that is a SUM counter and is defined in the same CONTROL FOOTING as this SUM clause, is summed before presentation of this CONTROL FOOTING. This counter updating is commonly called cross-footing. SUM counter operands are added to their respective SUM counters in the order in which they physically appear in the CONTROL FOOTING report group description, i.e., left to right within an elementary item entry and down the elementary item entries.

The UPON data-name option is required to obtain selective summation for a particular data item that is named as a SOURCE item in two or more DETAIL report groups. Identifier-2 and identifier-3 must be SOURCE data items in data-name. However, this compiler does not require that identifier-2, identifier-3, etc., be SOURCE data items in data-name. Data-name must be the name of a DETAIL report group.

The following is the chronology of summing events.

1. Cross-footing and counter rolling.
2. Execution of the USE BEFORE REPORTING procedure.
3. Presentation of the control footing if it is not a null group.

GENERATE Statement

4. SUM counter resetting unless an explicit RESET clause appears in the entry that defines the SUM counter.

TALLY may be used in place of an identifier in the SUM clause. It is treated as an elementary numeric data item.

Note: Undefined operands in the SUM clause are not diagnosed by the compiler, and no warning message is issued. Thus, when using the report writer, ensure that no operand is misspelled or that each operand is defined previously in the working storage, file, or linkage section or is the name of some other sum-counter.

VALUE: The VALUE clause causes the report data item to assume the specified value each time its report group is presented only if the elementary item entry does not contain a GROUP INDICATE clause. If the GROUP INDICATE clause is present and a given object time condition exists, the item will not assume the specified value (see GROUP INDICATE rules).

PROCEDURE DIVISION CONSIDERATIONS

To produce a report, the INITIATE, GENERATE, and TERMINATE statements must be specified in the Procedure Division. In addition, a USE BEFORE REPORTING declarative may be written in a Declarative Section of the Procedure Division. This option allows the programmer to manipulate or alter data immediately before it is printed.

GENERATE Statement

The GENERATE statement is used to produce a report.

Format
<u>GENERATE</u> identifier

Identifier is the name of either a DETAIL report group or an RD entry. If identifier is the name of a DETAIL report group, it can be made unique through qualification, using the associated report-name.

Detail Reporting

If identifier is the name of a DETAIL report group, the GENERATE statement does all the automatic operations within a Report Writer program and produces an actual output detail report group on the output medium. At least one DETAIL report group must be specified.

Summary Reporting

If identifier is the name of an RD entry, the GENERATE statement does all of the automatic operations of the Report Writer except producing any detail report group associated with the report. For summary reporting, a DETAIL report group need not be specified.

GENERATE Statement

In summary reporting, SUM counters are algebraically incremented in the same manner as for detail reporting. If more than one DETAIL report group is specified in a report, SUM counters are algebraically incremented as though consecutive GENERATE statements were issued for all the DETAIL report groups of the report. This consecutive summing takes place in the order of the physical appearance of the DETAIL report group descriptions. Even if there is more than one DETAIL report group within a report, only one test for control break is made for each GENERATE report-name. This test is made by the Report Writer prior to the summary reporting. After initiating a report and before terminating the same report, both detail reporting and summary reporting may be performed.

Operation of the GENERATE Statement

A GENERATE statement, implicitly in both detail and summary reporting, produces the following automatic operations (if defined):

1. Steps and tests the LINE COUNTER and/or PAGE COUNTER to produce appropriate PAGE FOOTING and/or PAGE HEADING report groups, after a line is printed.
2. Recognizes any specified control breaks to produce appropriate CONTROL FOOTING and/or CONTROL HEADING report groups.
3. Accumulates into the SUM counters all specified identifier(s). Resets the SUM counters.
4. Executes any specified routines defined by a USE statement before generation of the associated report group(s).

During the execution of the first GENERATE statement, the following report groups associated with the report (if specified) are produced in the order:

1. REPORT HEADING report group
2. PAGE HEADING report group
3. All CONTROL HEADING report groups in the order FINAL, major to minor
4. The DETAIL report group if specified in the GENERATE statement.

If a control break is recognized at the time of the execution of a GENERATE statement (other than the first that is executed for a report), all CONTROL FOOTING report groups specified for the report are produced from the minor report group, up to and including the report group specified for the identifier which caused the control break. Then, the CONTROL HEADING report group(s) specified for the report are produced, starting with the report group specified for the identifier that caused the control break, and continuing down to and ending with the minor report group. Then, the DETAIL report group specified in the GENERATE statement is produced.

Data is moved to the data item in the Report Group Description entry of the Report Section and is edited under control of the Report Writer according to the same rules for movement and editing as described for the MOVE statement (see "Procedure Division").

INITIATE Statement

The INITIATE statement begins the processing of a report.

Format
<u>INITIATE</u> report-name-1 [report-name-2] ...

Each report-name must be defined by a Report Description entry in the Report Section of the Data Division.

The INITIATE statement resets all data-name entries that contain SUM clauses associated with the report; the Report Writer controls for all the TYPE report groups that are associated with this report are set up in their respective order.

The PAGE-COUNTER register, if specified, is set to 1 (one) during the execution of the INITIATE statement. If a starting value other than 1 is desired, the programmer may reset this PAGE-COUNTER following the INITIATE statement.

The LINE-COUNTER register, if specified, is set to zero during the execution of the INITIATE statement.

The PRINT-SWITCH register is set to zero during the execution of the INITIATE statement.

The INITIATE statement does not open the file with which the report is associated; an OPEN statement for the file must be given by the user. The INITIATE statement performs Report Writer functions for individually described reports analogous to the input and/or output functions that the OPEN statement performs for individually described files.

A second INITIATE statement for a particular report-name may not be executed unless a TERMINATE statement has been executed for that report-name subsequent to the first INITIATE statement.

TERMINATE Statement

The TERMINATE statement completes the processing of a report.

Format
<u>TERMINATE</u> report-name-1 [report-name-2] ...

Each report-name given in a TERMINATE statement must be defined by an RD entry in the Data Division.

USE BEFORE REPORTING Declarative

The TERMINATE statement produces all the CONTROL FOOTING report groups associated with this report as though a control break had just occurred at the highest level, and completes the Report Writer functions for the named reports. The TERMINATE statement also produces the last REPORT FOOTING report group associated with this report.

Appropriate PAGE HEADING and/or PAGE FOOTING report groups are prepared in their respective order for the report description.

A second TERMINATE statement for a particular report may not be executed unless a second INITIATE statement has been executed for the report-name.

The TERMINATE statement does not close the file with which the report is associated; a CLOSE statement for the file must be given by the user. The TERMINATE statement performs Report Writer functions for individually described report programs analogous to the input/output functions that the CLOSE statement performs for individually described files.

If, at object time, no GENERATE statement is executed for a report, the TERMINATE statement of the report will not produce any report groups and will not perform any SUM processing.

SOURCE clauses used in CONTROL FOOTING FINAL or REPORT FOOTING report groups refer to the values of the items during the execution of the TERMINATE statement.

USE Sentence

The USE sentence specifies Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

Format
<u>USE BEFORE REPORTING</u> data-name.

A USE sentence, when present, must immediately follow a section header in the declaratives portion of the Procedure Division and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

Data-name represents a report group named in the Report Section of the Data Division. A data-name must not appear in more than one USE sentence. Data-name must be qualified by the report-name if data-name is not unique.

No Report Writer statement (GENERATE, INITIATE, or TERMINATE) may be written in a procedural paragraph(s) following the USE sentence in the declaratives portion.

The USE sentence itself is never executed; rather it defines the conditions calling for the execution of the USE procedures.

The designated procedures are executed by the Report Writer just before the named report group is produced, regardless of page or control break associations with report groups. The report group may be any type except DETAIL.

Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion, there must be no reference to procedure names that appear in the Declaratives Section, except that PERFORM statements may refer to a USE procedure or to the procedures associated with the USE procedure.

When the user wishes to suppress the printing of the specified report groups, the statement

```
MOVE 1 TO PRINT-SWITCH
```

is used in the USE BEFORE REPORTING declarative section. When this statement is encountered, only the specified report group is not printed; the statement must be written for each report group whose printing is to be suppressed.

Use of PRINT-SWITCH to suppress the printing of a report group implies that:

1. Nothing is printed
2. The LINE-COUNTER is not altered
3. The function of the NEXT GROUP clause, if one appears in the report group description, is nullified

SPECIAL REGISTERS: PAGE-COUNTER AND LINE-COUNTER

The fixed data-names, PAGE-COUNTER and LINE-COUNTER, are numeric counters automatically generated by the Report Writer based on the presence of specific entries; they do not require data description clauses. The description of these two counters is included here in order to explain their resultant effect on the overall report format.

PAGE-COUNTER

A PAGE-COUNTER is a counter generated by the Report Writer to be used as a source data item in order to present the page number on a report line. A PAGE-COUNTER is generated for a report by the Report Writer if a PAGE-LIMIT clause is specified in the RD entry of the report. The numeric counter is a 3-byte COMPUTATIONAL-3 item that is presented according to the PICTURE clause associated with the elementary item whose SOURCE is PAGE-COUNTER.

If more than one PAGE-COUNTER is given as a SOURCE data item within a given report, the number of numeric characters indicated by the PICTURE clauses must be identical. If more than one PAGE-COUNTER exists in the program, the user must qualify PAGE-COUNTER by the report name.

PAGE-COUNTER may be referred to in Report Section entries and in Procedure Division statements. After an INITIATE statement, PAGE-COUNTER contains one; if a starting value for PAGE-COUNTER other than one is desired, the programmer may change the contents of the PAGE-COUNTER by a Procedure Division statement after an INITIATE statement has been executed. PAGE-COUNTER is automatically incremented by one each time a page break is recognized by the Report Writer, after

LINE-COUNTER Special Register

the production of any PAGE FOOTING report group but before production of any PAGE HEADING report group.

LINE-COUNTER

A LINE-COUNTER is a counter used by the Report Writer to determine when a PAGE HEADING and/or a PAGE FOOTING report group is to be presented. One line counter is supplied for each report with a PAGE LIMIT(S) clause written in the Report Description entry (RD). The numeric counter is a 3-byte ~~COMPARE~~ item that is presented according to the PICTURE clause associated with the elementary item whose SOURCE is LINE-COUNTER.

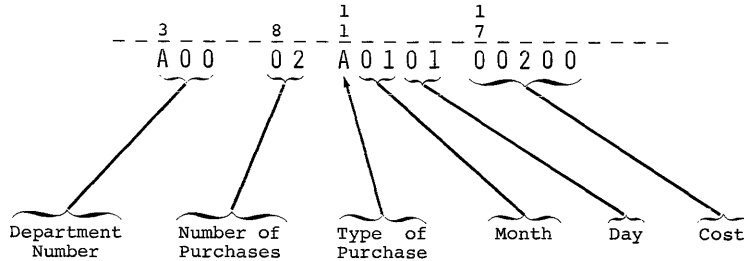
LINE-COUNTER may be referred to in Report Section entries and in Procedure Division statements. If more than one Report Description entry (RD) exists in the Report Section, the user must qualify LINE-COUNTER by the report-name. LINE-COUNTER is automatically tested and incremented by the Report Writer based on control specifications in the PAGE LIMIT(S) clause and values specified in the LINE NUMBER and NEXT GROUP clauses. After an INITIATE statement, LINE-COUNTER contains zero. Changing the value of LINE-COUNTER by Procedure Division statements may cause page format control to become unpredictable in the Report Writer.

The value of LINE-COUNTER during any Procedure Division test statement represents the number of the last line printed by the previously generated report group or represents the number of the last line skipped to by a previous NEXT GROUP specification.

In a USE BEFORE REPORTING, if no lines have been printed or skipped on the current page, LINE-COUNTER will contain zero. In all other cases, LINE-COUNTER represents the last line printed or skipped.

SAMPLE REPORT WRITER PROGRAM

Figure 46 illustrates a Report Writer source program. The records used in the report (i.e., input data) are shown after the STOP RUN card in the program. Using the first record as an example, the data fields are arranged in the following card format:



The decimal point in the cost field is assumed to be two places from the right.

```

000005 IDENTIFICATION DIVISION.
000010 PROGRAM-ID. ACME.
000015 REMARKS. THE REPORT WAS PRODUCED BY THE REPORT WRITER.
000020 ENVIRONMENT DIVISION.
000025 CONFIGURATION SECTION.
000030 SOURCE-COMPUTER. IBM-360-F50.
000035 OBJECT-COMPUTER. IBM-360-F50.
000040 INPUT-OUTPUT SECTION.
000045 FILE-CONTROL.
000050        SELECT INFILE ASSIGN TO SYS000-UT-2400-S.
000055        SELECT REPORT-FILE ASSIGN TO SYS001-UT-2400-S.
000060 DATA DIVISION.
000065 FILE SECTION.
000070 FD INFILE
000075        LABEL RECORDS ARE OMITTED
000080        DATA RECORD IS INPUT-RECORD.
000085 01 INPUT-RECORD.
000090        05 FILLER                PICTURE AA.
000095        05 DEPT                 PICTURE XXX.
000100        05 FILLER                PICTURE AA.
000105        05 NO-PURCHASES        PICTURE 99.
000110        05 FILLER                PICTURE A.
000115        05 TYPE-PURCHASE       PICTURE A.
000120        05 MONTH               PICTURE 99.
000125        05 DAY-1               PICTURE 99.
000130        05 FILLER               PICTURE A.
000135        05 COST                PICTURE 999V99.
000140        05 FILLER               PICTURE X(59).
000145 FD REPORT-FILE
000150        LABEL RECORDS ARE STANDARD
000151        RECORD CONTAINS 121 CHARACTERS
000155        REPORT IS EXPENSE-REPORT.
000160 WORKING-STORAGE SECTION.
000165 77 SAVED-MONTH                PICTURE 99 VALUE IS 0.
000175 77 CONTINUED                   PICTURE X(11) VALUE IS SPACE.
  
```

Figure 46. Sample Program Using the Report Writer Feature (Part 1 of 4)

```

000180 01 MONTH-TABLE-1.
000185 05 RECORD-MONTH.
000190 10 FILLER PICTURE A(9) VALUE IS "JANUARY ".
000195 10 FILLER PICTURE A(9) VALUE IS "FEBRUARY ".
000200 10 FILLER PICTURE A(9) VALUE IS "MARCH ".
000205 10 FILLER PICTURE A(9) VALUE IS "APRIL ".
000210 10 FILLER PICTURE A(9) VALUE IS "MAY ".
000215 10 FILLER PICTURE A(9) VALUE IS "JUNE ".
000220 10 FILLER PICTURE A(9) VALUE IS "JULY ".
000225 10 FILLER PICTURE A(9) VALUE IS "AUGUST ".
000230 10 FILLER PICTURE A(9) VALUE IS "SEPTEMBER".
000235 10 FILLER PICTURE A(9) VALUE IS "OCTOBER ".
000240 10 FILLER PICTURE A(9) VALUE IS "NOVEMBER ".
000245 10 FILLER PICTURE A(9) VALUE IS "DECEMBER ".
000250 05 RECORD-AREA REDEFINES RECORD-MONTH.
000255 10 MONTHNAME PICTURE A(9) OCCURS 12 TIMES.
000260 REPORT SECTION.
000265 RD EXPENSE-REPORT
000270 CONTROLS ARE FINAL MONTH DAY-1
000275 PAGE LIMIT IS 59 LINES
000280 HEADING 1
000285 FIRST DETAIL 9
000290 LAST DETAIL 48
000295 FOOTING 52.
000300 01 TYPE IS REPORT HEADING.
000305 05 LINE NUMBER IS 1
000310 COLUMN NUMBER IS 27
000315 PICTURE IS A(26)
000320 VALUE IS "ACME MANUFACTURING COMPANY".
000325 05 LINE NUMBER IS 3
000330 COLUMN NUMBER IS 26
000335 PICTURE IS A(29)
000340 VALUE IS "QUARTERLY EXPENDITURES REPORT".
000345 01 PAGE-HEAD
000350 TYPE IS PAGE HEADING.
000355 05 LINE NUMBER IS 5.
000360 10 COLUMN IS 30
000365 PICTURE IS A(9)
000370 SOURCE IS MONTHNAME (MONTH).
000375 10 COLUMN IS 39
000380 PICTURE IS A(12)
000385 VALUE IS "EXPENDITURES".
000390 10 COLUMN IS 52
000395 PICTURE IS X(11)
000400 SOURCE IS CONTINUED.
000405 05 LINE IS 7.
000410 10 COLUMN IS 2
000415 PICTURE IS X(35)
000420 VALUE IS "MONTH DAY DEPT NO-PURCHASES".
000425 10 COLUMN IS 40
000430 PICTURE IS X(33)
000435 VALUE IS "TYPE COST CUMULATIVE-COST".

```

Figure 46. Sample Program Using the Report Writer Feature (Part 2 of 4)

```

000440 01  DETAIL-LINE TYPE IS DETAIL LINE NUMBER IS PLUS 1.
000445 05  COLUMN IS 2 GROUP INDICATE PICTURE IS A(9)
000450      SOURCE IS MONTHNAME (MONTH).
000455 05  COLUMN IS 13 GROUP INDICATE PICTURE IS 99
000460      SOURCE IS DAY-1.
000465 05  COLUMN IS 19 PICTURE IS XXX SOURCE IS DEPT.
000470 05  COLUMN IS 31 PICTURE IS Z9 SOURCE IS NO-PURCHASES.
000475 05  COLUMN IS 42 PICTURE IS A SOURCE IS TYPE-PURCHASE.
000480 05  COLUMN IS 50 PICTURE IS ZZ9.99 SOURCE IS COST.
000485 01  TYPE IS CONTROL FOOTING DAY-1.
000490 05  LINE NUMBER IS PLUS 2.
000495 10  COLUMN 2 PICTURE X(22)
000500      VALUE "PURCHASES AND COST FOR".
000505 10  COLUMN 24 PICTURE Z9 SOURCE SAVED-MONTH.
000510 10  COLUMN 26 PICTURE X VALUE "-".
000515 10  COLUMN 27 PICTURE 99 SOURCE DAY-1.
000520 10  COLUMN 30 PICTURE ZZ9 SUM NO-PURCHASES.
000525 10  MIN -
000530      COLUMN 49 PICTURE $$$9.99 SUM COST.
000535 10  COLUMN 65 PICTURE $$$9.99 SUM COST.
000540      RESET ON FINAL.
000545 05  LINE PLUS 1 COLUMN 2 PICTURE X(71)
000550      VALUE ALL "*".
000555 01  TYPE CONTROL FOOTING MONTH
000560      LINE PLUS 1 NEXT GROUP NEXT PAGE.
000565 05  COLUMN 16 PICTURE A(14) VALUE "TOTAL COST FOR".
000570 05  COLUMN 31 PICTURE A(9)
000575      SOURCE MONTHNAME (MONTH).
000580 05  COLUMN 43 PICTURE AAA VALUE "WAS".
000585 05  INT
000590      COLUMN 48 PICTURE $$$9.99 SUM MIN..
000595 01  TYPE CONTROL FOOTING FINAL LINE NEXT PAGE.
000600 05  COLUMN 16 PICTURE A(26)
000605      VALUE "TOTAL COST FOR QUARTER WAS".
000610 05  COLUMN 45 PICTURE $$$9.99 SUM INT..
000615 01  TYPE PAGE FOOTING LINE 57.
000620 05  COLUMN 59 PICTURE X(12) VALUE "REPORT-PAGE-".
000625 05  COLUMN 71 PICTURE 99 SOURCE PAGE-COUNTER.
000630 01  TYPE REPORT FOOTING
000635      LINE PLUS 1 COLUMN 32 PICTURE A(13)
000640      VALUE "END OF REPORT".
000645  PROCEDURE DIVISION.
000650  DECLARATIVES.
000655  PAGE-HEAD-RTN SECTION.
000660      USE BEFORE REPORTING PAGE-HEAD.
000665  PAGE-HEAD-RTN-SWITCH.
000670      GO TO PAGE-HEAD-RTN-TEST.
000675  PAGE-HEAD-RTN-TEST.
000680      IF MONTH = SAVED-MONTH MOVE "(CONTINUED)" TO CONTINUED
000685      ELSE MOVE SPACES TO CONTINUED
000690      MOVE MONTH TO SAVED-MONTH.
000695      GO TO PAGE-HEAD-RTN-EXIT.
000697  PAGE-HEAD-RTN-ALTER.
000698      ALTER PAGE-HEAD-RTN-SWITCH TO PAGE-HEAD-RTN-SUPPRESS.
000700  PAGE-HEAD-RTN-SUPPRESS.
000705      MOVE 1 TO PRINT-SWITCH.
000710  PAGE-HEAD-RTN-EXIT.
000715      EXIT.
000720  END DECLARATIVES.

```

Figure 46. Sample Program Using the Report Writer Feature (Part 3 of 4)

```

000722  NON-DECLARATIVES SECTION.
000725  OPEN-FILES.  OPEN INPUT INFILE OUTPUT REPORT-FILE.
000730      INITIATE EXPENSE-REPORT.
000735  READATA.
000740      READ INFILE AT END GO TO COMPLETE.
000745      GENERATE DETAIL-LINE.
000760      GO TO READATA.
000765  COMPLETE.
000770      PERFORM PAGE-HEAD-RTN-ALTER.
000780      TERMINATE EXPENSE-REPORT.
000785      CLOSE INFILE REPORT-FILE.
000790      STOP RUN.
      .
      .
A00  02  A0101  00200
A02  01  A0101  00100
A02  02  C0101  01600
A01  02  B0102  00200
A04  10  A0102  01000
      .
      .
A01  06  C0329  04800
A03  20  E0331  06000

```

Figure 46. Sample Program Using the Report Writer Feature (Part 4 of 4)

Key Relating Report to Report Writer Source Program

In the key, the numbers enclosed in circles (for example, ①) relate the explanation below to the corresponding output line in Figure 47.

The six-digit numbers (for example, 000615) show the source statement from the program illustrated in Figure 46.

- ① is the REPORT HEADING resulting from source lines 000300-000340.
- ② is the PAGE HEADING resulting from source lines 000345-000435.
- ③ is the DETAIL line resulting from source lines 000440-000480 (note that since it is the first detail line after a control break, the fields defined with the GROUP INDICATE clause, lines 000445-000460, appear).
- ④ is a DETAIL line resulting from the same source lines as ③. In this case, however, the fields described as GROUP INDICATE do not appear (since the control break did not immediately precede the detail line).
- ⑤ is the CONTROL FOOTING (for DAY-1) resulting from source lines 000485-000550.
- ⑥ is the PAGE FOOTING resulting from source lines 000615-000625.
- ⑦ is the CONTROL FOOTING (for MONTH) resulting from source lines 000555-000575.

- ⑧ is the CONTROL FOOTING (for FINAL) resulting from source lines 000595-000610.

- ⑨ is the REPORTING FOOTING resulting from source lines 000630-000640.

Lines 000650-000715 of the example illustrate a use of USE BEFORE REPORTING. The effect of the source is that each time a new page is started, a test is made to determine whether the new page is being started because a change in MONTH has been recognized (the definition for the control footing for MONTH specifies NEXT GROUP NEXT PAGE) or because the physical limits of the page were exhausted. If a change in MONTH has been recognized, spaces are moved to the PAGE HEADING; if the physical limits of the page are exhausted, "(CONTINUED)" is moved to the PAGE HEADING.

①	ACME MANUFACTURING COMPANY						
	QUARTERLY EXPENDITURES REPORT						
②	JANUARY EXPENDITURES						
	MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST
③	JANUARY	01	A00	2	A	2.00	
			A02	1	A	1.00	
④			A02	2	C	16.00	
⑤	PURCHASES AND COST FOR 1-01				5	\$19.00	\$19.00

	JANUARY	02	A01	2	B	2.00	
			A04	10	A	10.00	
			A04	10	C	80.00	
	PURCHASES AND COST FOR 1-02				22	\$92.00	\$111.00

	JANUARY	05	A01	2	B	2.00	
	PURCHASES AND COST FOR 1-05				2	\$2.00	\$113.00

	JANUARY	08	A01	10	A	10.00	
			A01	8	B	12.48	
			A01	20	D	38.40	
	PURCHASES AND COST FOR 1-08				38	\$60.88	\$173.88

	JANUARY	13	A00	4	B	6.24	
			A00	1	C	8.00	
	PURCHASES AND COST FOR 1-13				5	\$14.24	\$188.12

	JANUARY	15	A00	10	D	19.20	
			A02	1	C	8.00	
	PURCHASES AND COST FOR 1-15				11	\$27.20	\$215.32

	JANUARY	21	A03	10	E	30.00	
			A03	10	F	25.00	
			A03	10	G	50.00	
	PURCHASES AND COST FOR 1-21				30	\$105.00	\$320.32

	JANUARY	23	A00	5	A	5.00	
	PURCHASES AND COST FOR 1-23				5	\$5.00	\$325.32

⑥	REPORT-PAGE-01						

Figure 47. Report Produced by Report Writer Feature (Part 1 of 5)

② JANUARY EXPENDITURES (CONTINUED)							
MONTH	DAY	DEPT	NO-PURCHASES	TYPE	CGST	CUMULATIVE-COST	
③ JANUARY	26	A04	5	A	5.00		
		A04	5	B	7.80		
④ PURCHASES AND COST FOR 1-26			10		\$12.80	\$338.12	

⑤ JANUARY	27	A00	6	B	9.36		
		A00	15	C	120.00		
PURCHASES AND COST FOR 1-27			21		\$129.36	\$467.48	

JANUARY	30	A00	2	B	3.12		
		A02	10	A	10.00		
		A02	1	C	8.00		
		A04	15	B	23.40		
		A04	10	C	80.00		
PURCHASES AND COST FOR 1-30			38		\$124.52	\$592.00	

JANUARY	31	A00	1	A	1.00		
		A04	6	A	6.00		
PURCHASES AND COST FOR 1-31			7		\$7.00	\$599.00	

⑦ TOTAL COST FOR JANUARY					WAS	\$599.00	

⑥ REPORT-PAGE-02

Figure 47. Report Produced by Report Writer Feature (Part 2 of 5)

Report Writer -- Sample Program

2 ----- FEBRUARY EXPENDITURES							
MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST	
3	FEBRUARY	15	A02	10	A	10.00	
			A02	2	B	3.12	
			A02	1	C	8.00	
			A03	15	G	75.00	
			A04	5	B	7.80	
			A05	8	A	8.00	
			A05	5	C	40.00	
PURCHASES AND COST FOR 2-15					46	\$151.92	\$750.92

4	FEBRUARY	16	A02	2	C	16.00	
			A06	10	A	10.00	
			A07	10	A	10.00	
			A07	10	F	25.00	
PURCHASES AND COST FOR 2-16					32	\$61.00	\$811.92

5	FEBRUARY	17	A07	10	E	30.00	
			A07	10	G	50.00	
PURCHASES AND COST FOR 2-17					20	\$80.00	\$891.92

6	FEBRUARY	21	A06	20	A	20.00	
			A06	20	B	31.20	
			A06	20	C	160.00	
			A06	20	D	38.40	
			A06	20	E	60.00	
			A06	20	F	50.00	
			A06	20	G	100.00	
PURCHASES AND COST FOR 2-21					140	\$459.60	\$1351.52

7	FEBRUARY	27	A01	21	D	40.32	
		PURCHASES AND COST FOR 2-27					21

8	FEBRUARY	28	A02	3	B	4.68	
			A02	5	C	40.00	
			A03	15	E	45.00	
PURCHASES AND COST FOR 2-28					23	\$89.68	\$1481.52

TOTAL COST FOR FEBRUARY					WAS	\$882.52	

6 ----- REPORT-PAGE-03

Figure 47. Report Produced by Report Writer Feature (Part 3 of 5)

②		MARCH EXPENDITURES						
MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST		
③	MARCH	01	A02	5	A	5.00		
			A02	1	C	8.00		
④			A03	25	G	125.00		
⑤	PURCHASES AND COST FOR 3-01			31		\$138.00	\$1619.52	

	MARCH	06	A02	5	A	5.00		
	PURCHASES AND COST FOR 3-06			5		\$5.00	\$1624.52	

	MARCH	07	A02	5	A	5.00		
	PURCHASES AND COST FOR 3-07			5		\$5.00	\$1629.52	

	MARCH	13	A02	10	A	10.00		
	PURCHASES AND COST FOR 3-13			10		\$10.00	\$1639.52	

	MARCH	15	A01	21	A	21.00		
			A02	1	A	1.00		
			A03	15	F	37.50		
			A06	5	F	15.00		
			A06	5	F	12.50		
	PURCHASES AND COST FOR 3-15			47		\$87.00	\$1726.52	

	MARCH	20	A03	15	E	45.00		
	PURCHASES AND COST FOR 3-20			15		\$45.00	\$1771.52	

	MARCH	21	A02	15	A	15.00		
			A02	15	F	37.50		
	PURCHASES AND COST FOR 3-21			30		\$52.50	\$1824.02	

	MARCH	23	A02	2	A	2.00		
	PURCHASES AND COST FOR 3-23			2		\$2.00	\$1826.02	

	MARCH	25	A03	30	F	75.00		
	PURCHASES AND COST FOR 3-25			30		\$75.00	\$1901.02	

Figure 47. Report Produced by Report Writer Feature (Part 4 of 5)

Report Writer -- Sample Program

2		MARCH EXPENDITURES (CONTINUED)					
MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST	
3	MARCH	26	A02	1	A	1.00	
5	PURCHASES AND COST FOR 3-26			1		\$1.00	\$1902.02

	MARCH	29	A01	6	C	48.00	
	PURCHASES AND COST FOR 3-29			6		\$48.00	\$1950.02

	MARCH	31	A03	20	E	60.00	
	PURCHASES AND COST FOR 3-31			20		\$60.00	\$2010.02

7	TOTAL COST FOR MARCH				WAS	\$528.50	
6	REPORT-PAGE-05						
~~~~~							
8	TOTAL COST FOR QUARTER WAS \$2010.02						
6	REPORT-PAGE-06						
9	END OF REPORT						
~~~~~							

Figure 47. Report Produced by Report Writer Feature (Part 5 of 5)

TABLE HANDLING FEATURE

The Table Handling feature enables the programmer to process tables or lists of repeated data conveniently. A table may have up to three dimensions, i.e., three levels of subscripting or indexing can be handled. Such a case exists when a group item described with an OCCURS clause contains another group item with an OCCURS clause, which in turn contains an item with an OCCURS clause. To make reference to any element within such a table, each level must be subscripted or indexed.

SUBSCRIPTING

Subscripts are used only to refer to an individual element within a list or table of elements that have not been assigned individual data-names.

Format
data-name (subscript[, subscript][, subscript])

The subscript, or set of subscripts, that identifies the table element is enclosed in parentheses immediately following the space that terminates data-name, which is the name of the table element. When more than one subscript appears within a pair of parentheses, each subscript must be separated from the next by a comma followed by a space. ~~However, this compiler allows the comma to be omitted.~~ No space may appear between the left parenthesis and the leftmost subscript or between the rightmost subscript and the right parenthesis. To identify an element in the table named SALARY by the set of subscripts YEAR, MONTH, and WEEK, the programmer would write:

```
SALARY (YEAR, MONTH, WEEK).
```

The subscript can be represented by a numeric literal that is a positive integer, by the special register TALLY, or by a data-name. Restrictions on the use of a data-name as a subscript are:

1. Data-name must be a numeric elementary item that represents a positive integer.
2. The name itself may be qualified, but not subscripted.

The subscript may contain a sign, but the lowest permissible subscript value is 1. Hence, the use of zero or a negative subscript is not permitted. The highest permissible subscript value in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

Qualification may be used in conjunction with subscripting, in which case OF or IN follows the data-name being subscripted.

Subscripting and Indexing

Format

$$\text{data-name} \left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-1} [\left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\} \text{data-name-2}] \dots$$

(subscript[, subscript][, subscript])

Note: Data-name is the item being subscripted, not data-name-1. That is, in the statement SALARY OF EMPLOYEE-RECORD (YEAR, MONTH, WEEK), the data item SALARY is subscripted by YEAR, MONTH, and WEEK.

INDEXING

References can be made to individual elements within a table of elements by specifying indexing for that reference.

An index is assigned to a given level of a table by using an INDEXED BY clause in the definition of the table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index. An index-name must be initialized by a SET or PERFORM statement before it is used in a table reference. An index may be modified only by a SET, SEARCH, or PERFORM statement.

Format

$$\text{data-name} (\text{index-name} [\left\{ \begin{array}{c} + \\ - \end{array} \right\} \text{integer}]$$
$$[, \text{index-name} [\left\{ \begin{array}{c} + \\ - \end{array} \right\} \text{integer}]][, \text{index-name} [\left\{ \begin{array}{c} + \\ - \end{array} \right\} \text{integer}]])$$

Direct indexing is specified by using an index-name in the form of a subscript. For example,

ELEMENT (PRIME-INDEX)

Relative indexing is specified when the terminal space of the data-name is followed by a parenthesized group of items: the index-name, followed by a space, followed by one of the operators + or -, followed by another space, followed by an unsigned integral numeric literal. For example,

ELEMENT (PRIME-INDEX + 5)

Qualification may be used in conjunction with indexing, in which case OF or IN follows the data-name being indexed.

Format									
data-name	$\left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\}$	data-name-1	[$\left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\}$	data-name-2]	...			
		(index-name	[$\left\{ \begin{array}{c} + \\ - \end{array} \right\}$	integer]	[, index-name	[$\left\{ \begin{array}{c} + \\ - \end{array} \right\}$	integer]]
		[, index-name	[$\left\{ \begin{array}{c} + \\ - \end{array} \right\}$	integer]]				

Note: Data items described by the USAGE IS INDEX clause permit storage of the values of index-names as data without conversion. Such items are called index data items.

RESTRICTIONS ON INDEXING, SUBSCRIPTING, AND QUALIFICATION

Tables may have one, two, or three dimensions. Therefore, references to an element in a table may require up to three subscripts or indexes.

1. A data-name must not be subscripted or indexed when the data-name is itself being used as an index, subscript, or qualifier.
2. When qualification, subscripting, or indexing are required for a given data item, the indexes or subscripts are specified after all necessary qualification is given.
3. Subscripting and indexing must not be used together in a single reference.
4. Wherever subscripting is not permitted, indexing is not permitted.
5. The commas shown in the formats for indexes and subscripts are required.

EXAMPLE OF SUBSCRIPTING AND INDEXING

For a table with three levels of indexing, the following Data Division entries would result in a storage layout as shown in Figure 48.

```

01 PARTY-TABLE REDEFINES TABLE.
   02 PARTY-CODE OCCURS 3 TIMES INDEXED BY PARTY.
     03 AGE-CODE OCCURS 3 TIMES INDEXED BY AGE.
       04 M-F-INFO OCCURS 2 TIMES INDEXED BY M-F
         PICTURE 9(7)V9 USAGE DISPLAY.
```

PARTY-TABLE contains three levels of indexing. Reference to elementary items within PARTY-TABLE is made by use of a name that is subscripted or indexed. A typical Procedure Division statement might be:

```
MOVE M-F-INFO (PARTY, AGE, M-F) TO M-F-RECORD.
```

In order to use the Table Handling feature, the programmer must provide certain information in the Data Division and Procedure Division of the program.

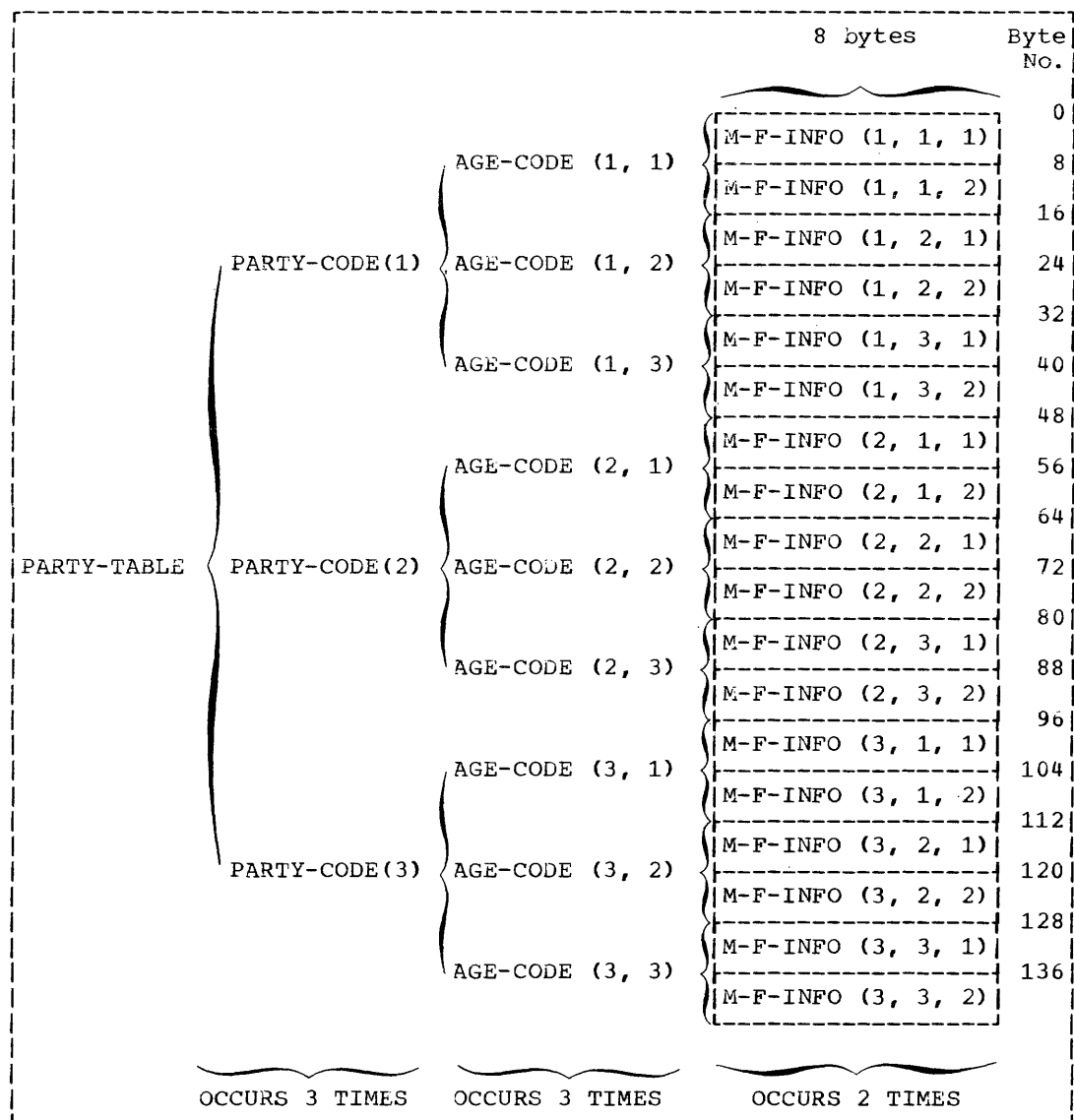


Figure 48. Storage Layout for PARTY-TABLE

DATA DIVISION CONSIDERATIONS FOR TABLE HANDLING

The OCCURS and USAGE clauses are included as part of the record description entries in a program utilizing the Table Handling feature.

OCCURS Clause

The OCCURS clause eliminates the need for separate entries for repeated data, since it indicates the number of times a series of records with identical format is repeated. It also supplies information required for the application of subscripts or indexes.

The OCCURS clause has three formats.

```

Format 1
-----
OCCURS integer-2 TIMES
  [ { ASCENDING }
    { DESCENDING } KEY IS data-name-2 [data-name-3] ... ] ...
  [ INDEXED BY index-name-1 [index-name-2] ... ]

```

```

Format 2
-----
OCCURS integer-1 TO integer-2 TIMES [ DEPENDING ON data-name-1 ]
  [ { ASCENDING }
    { DESCENDING } KEY IS data-name-2 [data-name-3] ... ] ...
  [ INDEXED BY index-name-1 [index-name-2] ... ]

```

```

Format 3
-----
OCCURS integer-2 TIMES [ DEPENDING ON data-name-1 ]
  [ { ASCENDING }
    { DESCENDING } KEY IS data-name-2 [data-name-3] ... ] ...
  [ INDEXED BY index-name-1 [index-name-2] ... ]

```

The other data description clauses associated with an entry whose description includes an OCCURS clause apply to each occurrence of the item described.

Since three subscripts or indexes are allowed, three nested levels of the OCCURS clause are allowed. That is, 3-dimensional tables can be specified. No table may be longer than 32,767 bytes in length, except for fixed-length tables in the Working-Storage Section or Linkage Section, which may be as long as 131,071 bytes.

The subject of an OCCURS clause is the data-name of the entry that contains this OCCURS clause. The subject of an OCCURS clause must be subscripted or indexed whenever reference is made to it in any statement other than SEARCH. The subject of an OCCURS clause must not be longer than 32,767 bytes.

When subscripted, the subject refers to one occurrence within the table. When not subscripted (permissible only in the SEARCH statement), the subject represents the entire table element.

OCCURS Clause

The OCCURS clause may not be specified in a data description entry that:

1. has a level-01 or level-77 number
2. describes an item whose size is variable

(The size of an item is variable if the data description of any subordinate item within it contains an OCCURS DEPENDING ON clause--that is, an OCCURS clause with the DEPENDING ON option.)

However, this compiler allows the size of any subordinate item to be variable -- that is, to contain an OCCURS DEPENDING ON clause.

Except for condition-name entries, a record description entry that contains an OCCURS clause must not also contain a VALUE clause.

Within a given record description, the VALUE clause must not be used in a data description entry that is subsequent to a data description entry that contains an OCCURS DEPENDING ON clause.

In the discussion that follows, the term "computational" refers to COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 data items.

When a computational elementary item specifies both the OCCURS and SYNCHRONIZED clauses, any necessary slack bytes for each occurrence of the item are added by the compiler. When a group item specifies the OCCURS clause and also contains SYNCHRONIZED computational elementary items, any necessary slack bytes for each occurrence of the group are added by the compiler, as well as the necessary slack bytes for the computational items (see "Slack Bytes" in "Data Division" for a complete discussion).

In Format 1, integer-2 represents the exact number of occurrences. In this case, integer-2 must be greater than zero.

DEPENDING ON OPTION: In Format 2 and Format 3, the DEPENDING ON option is used. This indicates that the subject of this entry has a variable number of occurrences. This does not mean that the length of the subject is variable, but rather that the number of times the subject may be repeated is variable, the number of times being controlled by the value of data-name-1 at object time.

In Format 2, integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. Integer-1 may be zero or any positive integer. Integer-2 must be greater than zero, and also greater than integer-1. Integer-2 must be less than 32,768. The value of data-name-1 must not exceed integer-2.

In Format 3, integer-2 represents the maximum number of occurrences, and it must be greater than zero and less than 32,768. The value of data-name-1 must not exceed integer-2.

Data-name-1, the object of the DEPENDING ON option:

- Must be described as a positive integer
- Must not exceed integer-2 in value
- May be qualified, when necessary
- Must not be subscripted (that is, must not itself be the subject of, or an entry within, a table)
- Must, if it appears in the same record as the table it controls, appear before the variable portion of the record

If the value of data-name-1 is reduced, the contents of data items whose occurrence numbers exceed the new value of data-name-1 become unpredictable.

Unused character positions resulting from the DEPENDING ON option will not appear on external media.

The DEPENDING ON option is required only when the last occurrence of the subject cannot otherwise be determined.

Any Data Division entry that contains an OCCURS DEPENDING ON clause, or which has subordinate to it an entry that contains an OCCURS DEPENDING ON clause, cannot be the object of a REDEFINES clause.

KEY OPTION: The KEY option is used in conjunction with the INDEXED BY option in the execution of a SEARCH ALL statement. The KEY option is used to indicate that the repeated data is arranged in ASCENDING or in DESCENDING order, according to the values contained in data-name-2, data-name-3, etc.

Data-name-2 must be either the name of the entry containing an OCCURS clause, or it must be an entry subordinate to the entry containing the OCCURS clause. If data-name-2 is the subject of this table entry, it is the only key that may be specified for this table. If data-name-2 is not the subject of this table entry, all the keys identified by data-name-2, data-name-3, etc.;

- Must be subordinate to the subject of the table entry itself
- Must not be subordinate to any other entry that contains an OCCURS clause
- Must not themselves contain an OCCURS clause

When the KEY option is specified, the following rules apply:

- Keys must be listed in descending order of significance.
- The total number of keys for a given table element must not exceed 12.
- The sum of the lengths of all keys associated with one table element must not exceed 256 bytes.
- A key may have the following usages: DISPLAY, COMPUTATIONAL-3, or COMPUTATIONAL.

OCCURS Clause

When subordinate entries within the table are variable in length, the following rule also applies:

- Any key in a table element must be at a fixed displacement from the beginning of that element (that is, if a table element is of variable length, then the keys must precede the variable portion).

The following example shows a violation of the last preceding rule:

```
WORKING-STORAGE SECTION.  
77 CURRENT-WEEK                PICTURE 99.  
01 TABLE-RECORD.  
  02 EMPLOYEE-TABLE OCCURS 100 TIMES  
    ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO  
    INDEXED BY A, B.  
  03 WEEK-RECORD OCCURS 1 TO 52 TIMES  
    DEPENDING ON CURRENT-WEEK  
    INDEXED BY C.  
    04 WEEK-NO                PIC 99.  
    04 AUTHORIZED-ABSENCES    PIC 9.  
    04 UNAUTHORIZED-ABSENCES PIC 9.  
    04 LATENESSES            PIC 9.  
  03 EMPLOYEE-NAME           PIC X(20).  
  03 EMPLOYEE-NO             PIC 9(6).  
  03 WAGE-RATE               PIC 9999V99.  
  .  
  .  
  .
```

WAGE-RATE and EMPLOYEE-NO are invalid as keys, since they are placed after the variable portion of the table.

The following example of the KEY option is correct:

```
WORKING-STORAGE SECTION.  
77 CURRENT-WEEK                PICTURE 99.  
01 TABLE-RECORD.  
  02 EMPLOYEE-TABLE OCCURS 100 TIMES  
    ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO  
    INDEXED BY A, B.  
  03 EMPLOYEE-NAME           PIC X(20).  
  03 EMPLOYEE-NO             PIC 9(6).  
  03 WAGE-RATE               PIC 9999V99.  
  03 WEEK-RECORD OCCURS 1 TO 52 TIMES  
    DEPENDING ON CURRENT-WEEK  
    INDEXED BY C.  
    04 WEEK-NO                PIC 99.  
    04 AUTHORIZED-ABSENCES    PIC 9.  
    04 UNAUTHORIZED-ABSENCES PIC 9.  
    04 LATENESSES            PIC 9.
```

The keys WAGE-RATE and EMPLOYEE-NO both appear at a fixed displacement from the beginning of the table element EMPLOYEE-TABLE.

INDEXED BY OPTION: The INDEXED BY option is required if the subject of this entry (the data-name described by the OCCURS clause, or an item within this data-name, if it is a group item) is to be referred to by indexing. The index-name(s) identified by this clause is not defined elsewhere in the program, since its allocation and format are dependent on the system, and, not being data, cannot be associated with any data hierarchy.

The number of index-names for a Data Division entry must not exceed 12.

An index-name must be initialized through a SET or PERFORM statement before it is used.

Each index-name is a fullword in length and contains a binary value that represents an actual displacement from the beginning of the table that corresponds to an occurrence number in the table. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by this index-name.

For example, if the programmer writes

```
A OCCURS 15 TIMES INDEXED BY Z PICTURE IS X(10)
```

on the fifth occurrence of A, the binary value contained in Z will be:

$$Z = (5 - 1) * 10 = 40$$

Note that, for a table entry of variable length, the value contained in the index-name entry will become invalid when the table entry length is changed, unless the user issues a new SET statement to correct the value contained in the index-name.

The following example of the setting of values in index-name is incorrect:

```

      .
      .
      .
DATA DIVISION.
      .
      .
      .
77  E PICTURE S9(5) COMP SYNC.
01  ...
      02  A OCCURS 10 INDEXED BY IND-1...
          03  B OCCURS 10 DEPENDING ON E INDEXED BY IND-2...
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
      MOVE 8 TO E
      SET IND-1 TO 3
      SEARCH A ...
      .
      .
      .
      MOVE 10 TO E
      SEARCH A ...

```

(Moving 10 to E changes the length of the table entry A, so that IND-1 now contains an invalid value.)

OCCURS Clause

The following example of the setting of values in index-name is correct:

```
      .  
      .  
      .  
DATA DIVISION.  
      .  
      .  
      .  
77  E PICTURE S9(5) COMP SYNC.  
77  D PICTURE S9(5) COMP SYNC.  
01  ...  
    02  A OCCURS 10 INDEXED BY IND-1...  
      03  B OCCURS 10 DEPENDING ON E INDEXED BY IND-2...  
      .  
      .  
      .  
PROCEDURE DIVISION.  
      .  
      .  
      .  
      MOVE 8 TO E  
      SET IND-1 TO 3  
      SET D TO IND-1  
      SEARCH A ...  
      .  
      .  
      .  
      MOVE 10 TO E  
      SET IND-1 TO D  
      SEARCH A ...  
      .  
      .  
      .
```

(Here the user has saved the occurrence number in D, and then later reset IND-1 to obtain the corrected value.)

There are two types of indexing: Direct Indexing and Relative Indexing.

Direct Indexing: If a data-name is used in the procedure text with index-names, the data-name itself must be the subject of an INDEXED BY option, or be subordinate to a group(s) that is the subject of the INDEXED BY option.

In the following example

```
A (INDEX-1, INDEX-2, INDEX-3)
```

implies that A belongs to a structure with three levels of OCCURS options, each with an INDEXED BY option. However, if data-name (A, in this example) belongs to an OCCURS structure that does not use the INDEXED BY option, this compiler accepts the specification of index-names (in this example INDEX-1, INDEX-2, INDEX-3) and assumes the user has set them to values that correspond to the occurrence number he wishes to reference.

Relative Indexing: The index-name is followed by a space, followed by one of the operators + or -, followed by another space, followed by an unsigned numeric literal. The numeric literal is considered to be an occurrence number, and is converted to an index value before being added to, or subtracted from, the corresponding index-name.

Given the following example:

A (Z + 1, J + 3, K + 4)

where:

table element indexed by Z has an entry length of 100

table element indexed by J has an entry length of 10

table element indexed by K has an entry length of 2

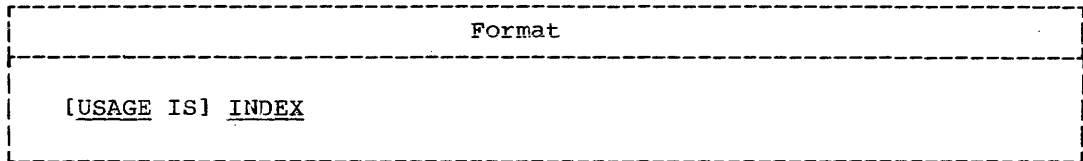
the resulting address will be computed as follows:

$$(\text{ADDRESS of A}) + Z + \boxed{100 * 1} + J + \boxed{10 * 3} + K + \boxed{4 * 2}$$

conversion of integers
to index values

USAGE IS INDEX Clause

The USAGE IS INDEX clause is used to specify the format of a data item stored internally.



The USAGE IS INDEX clause allows the programmer to specify index data items.

An index data item is an elementary item (not necessarily connected with any table) that can be used to save index-name values for future reference. An index data item must be assigned an index-name value (i.e., (occurrence number - 1) * entry length) through the SET statement. Such a value corresponds to an occurrence number in a table.

The USAGE IS INDEX clause may be written at any level. If a group item is described with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item can be referred to directly only in a SEARCH or SET statement or in a relation condition. An index data item can be part of a group which is referred to in a MOVE or an input/output

Table Handling -- Relation Condition

statement. When such operations are executed, however, there is no conversion of the contents of the index data item.

An index data item cannot be a conditional variable.

The SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause. However, this compiler allows the use of SYNCHRONIZED when USAGE IS INDEX to obtain efficient use of the item.

PROCEDURE DIVISION CONSIDERATIONS FOR TABLE HANDLING

The SEARCH and the SET statements may be used to facilitate table handling. In addition, there are special rules involving Table Handling elements when they are used in relation conditions.

Relation Conditions

Comparisons involving index-names and/or index data items conform to the following rules:

1. The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
2. In the comparison of an index-name with a data item (other than an index data item), or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
3. In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion.

Any other comparison involving an index data item is illegal.

Figure 49 gives permissible comparisons for index-names and index data items.

First Operand \ Second Operand	Index-name	Index Data Item	Data-name (numeric integer only)	Numeric literal (integer only)
Index-name	compare occurrence number	compare without conversion	compare occurrence number with data-name	compare occurrence number with literal
Index Data Item	compare without conversion	compare without conversion	illegal	illegal
Data-name (numeric integer only)	compare occurrence number with data-name	illegal	See Figure 23 for Permissible Comparisons	
Numeric literal (integer only)	compare occurrence number with literal	illegal		

Figure 49. Index-names and Index Data Items -- Permissible Comparisons

SEARCH Statement

The SEARCH statement is used to search a table for an element that satisfies a specified condition, and to adjust the value of the associated index-name to the occurrence number corresponding to that table element.

Format 1	
<u>SEARCH</u> identifier-1 [<u>VARYING</u> { index-name-1 }]	{ identifier-2 }
[<u>AT</u> <u>END</u> imperative-statement-1]	
<u>WHEN</u> condition-1	{ imperative-statement-2 } { <u>NEXT SENTENCE</u> }
[<u>WHEN</u> condition-2	{ imperative-statement-3 }]...
	{ <u>NEXT SENTENCE</u> }

SEARCH Statement

```

Format 2

SEARCH ALL identifier-1 [AT END imperative-statement-1]

      WHEN condition-1      { imperative-statement-2 }
                           { NEXT SENTENCE           }

```

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY option. Identifier-1 must not be described as a floating-point item.

When written in the SEARCH statement, identifier-1 must refer to all occurrences within one level of a table; that is, it must not be subscripted or indexed.

Identifier-1 can be a data item subordinate to a data item that contains an OCCURS clause, thus providing for a two or three dimensional table. An index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Execution of a SEARCH statement causes modification only of the setting of the index-name associated with identifier-1 (and, if present, of index-name-1 or identifier-2). Therefore, to search an entire two or three dimensional table, it is necessary to execute a SEARCH statement several times; prior to each execution, SET statements must be executed to adjust the associated index-names to their appropriate settings.

In the AT END and WHEN options, if any of the specified imperative statement(s) do not terminate with a GO TO statement, control passes to the next sentence after execution of the imperative statement.

Format 1 Considerations -- Identifier-2, when specified, must be described as an index data item, or it must be a fixed-point numeric elementary item described as an integer. When an occurrence number is incremented, identifier-2 is simultaneously incremented by the same amount.

Condition-1, condition-2, etc., may be any condition, as follows:

- relation condition
- class condition
- condition-name condition
- sign condition
- switch-status condition
- (condition)
- [NOT] { AND } condition
- { OR }

(See Conditions section of "Procedure Division.")

Upon the execution of a SEARCH statement, a serial search takes place, starting with the current index setting.

If, at the start of the SEARCH, the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number for identifier-1, the following actions take place:

1. The condition(s) in the WHEN option are evaluated in the order they are written.

2. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to reference the next table element, and step 1 is repeated.
3. if, upon evaluation, one of the WHEN conditions is satisfied, the search terminates immediately, and the imperative-statement associated with that condition is executed. The index-name points to the table element that satisfied the condition.
4. If the end of the table is reached without the WHEN condition being satisfied, the search terminates as described in the next paragraph.

If at the start of the search, the value of the index-name associated with identifier-1 is greater than the highest permissible occurrence number for identifier-1, the search is terminated immediately, and if the AT END option is specified, imperative-statement-1 is executed. If this option is omitted, control passes to the next sentence.

When the VARYING index-name-1 option is not specified, the index used for the search is the first (or only) index-name associated with identifier-1.

When the VARYING index-name-1 option is specified, one of the following applies:

- If index-name-1 is one of the indexes for identifier-1, index-name-1 is used for the search. Otherwise, the first (or only) index-name for identifier-1 is used.
- If index-name-1 is an index for another table entry, then when the index-name for identifier-1 is incremented to represent the next occurrence of the table, index-name-1 is simultaneously incremented to represent the next occurrence of the table it indexes.

A flowchart of the Format 1 SEARCH operation containing two WHEN options is shown in Figure 50.

Format 2 Considerations -- The first index-name assigned to identifier-1 will be used for the search.

The description of identifier-1 must contain the KEY option in its OCCURS clause.

Condition-1 must consist of one of the following:

- A relation condition incorporating the EQUALS, EQUAL TO, or equal sign (=) relation. Either the subject or the object (but not both) of the relation condition must consist solely of one of the data-names that appear in the KEY clause of identifier-1.
- A condition-name condition in which the VALUE clause describing the condition-name consists of a single literal only. The conditional variable associated with the condition-name must be one of the data-names that appear in the KEY clause of identifier-1.
- A compound condition formed from simple conditions of the types described above, with AND as the only connective.

Any data-name that appears in the KEY clause of identifier-1 may be tested in condition-1. However, all data-names in the KEY clause preceding the one to be tested must also be so tested in condition-1. No other tests may be made in condition-1.

For example, if the following table were defined in the Data Division:

```

77 VALUE-1 PICTURE 99.
      .
      .
      .
02 A OCCURS 10 TIMES ASCENDING KEY IS KEY1, KEY2, KEY3, KEY4
   INDEXED BY I.
   03 KEY1 PICTURE 9.
   03 KEY2 PICTURE 99.
   03 KEY3 PICTURE 9.
   03 KEY4 PICTURE 9.
   88 BLUE VALUE 1.
      .
      .
      .

```

in the Procedure Division, valid WHEN phrases could be:

```

WHEN KEY1 (I) = 3 AND KEY2 (I) = 10 AND KEY3 (I) = 5 ...

WHEN KEY1 (I) = 3 AND KEY2 (I) = VALUE-1
   AND KEY3 (I) = 5 AND BLUE (I) ...

```

During execution of a Format 2 SEARCH statement, a binary search takes place; the setting of index-name is varied during the search so that at no time is it less than the value that corresponds to the first element of the table, nor is it ever greater than the value that corresponds to the last element of the table. If condition-1 cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 when the AT END option appears, or to the next sentence when this clause does not appear. In either case, the final setting of the index is not predictable. If the index indicates an occurrence that allows condition-1 to be satisfied, control passes to imperative-statement-2.

SET Statement

SET Statement

The SET statement establishes reference points for table handling operations by setting index-names to values associated with table elements. The SET statement must be used when initializing index-name values before execution of a SEARCH statement; it may also be used to transfer values between index-names and other elementary data items.

```
Format 1
-----
SET { index-name-1 [index-name-2]... } TO { index-name-3 }
    { identifier-1 [identifier-2]... }   { identifier-3 }
                                         { literal-1 }
```

```
Format 2
-----
SET index-name-4 [index-name-5] ... { UP BY } { identifier-4 }
                                     { DOWN BY } { literal-2 }
```

All identifiers must name either index data items or fixed-point numeric elementary items described as integers; however, identifier-4 must not name an index data item. When a literal is used, it must be a positive integer or zero. Index-names are related to a given table through the INDEXED BY option of the OCCURS clause; when index-names are specified in the INDEXED BY option, they are automatically defined.

All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.

Format 1 Considerations -- When the SET statement is executed, one of the following actions occurs:

1. Index-name-1 is converted to a value that corresponds to the same table element to which either index-name-3, identifier-3, or literal-1 corresponds. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place. To be valid, the resultant value of index-name must correspond to the occurrence number of an element in the associated table.
2. If identifier-1 is an index data item, it is set equal to either the contents of index-name-3 or identifier-3, where identifier-3 is also an index data item. Literal-1 cannot be used in this case.
3. If identifier-1 is not an index data item, it is set to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor literal-1 can be used in this case.

Format 2 Considerations -- When the SET statement is executed, the contents of index-name-4 (and index-name-5, etc., if present) are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of literal-2 or identifier-4.

SAMPLE TABLE HANDLING PROGRAM

The program in Figure 51 illustrates the Table Handling feature, including the use of indexing, of the SET statement, and of the SEARCH statement (including the VARYING option and the SEARCH ALL format).

The census bureau uses the program to compare:

1. The number of births and deaths that occurred in any one of the 50 states in any one of the past 20 years with
2. The total number of births and deaths that occurred in the same state over the entire 20-year period

The input file, INCARDS, contains the specific information upon which the search of the table is to be conducted. INCARDS is formatted as follows:

STATE-NAME a 4-character alphabetic abbreviation of the state name
 M-F-CODE 1 = male; 2 = female
 YEARCODE a 4-digit field in the range 1950 through 1969

A typical run might determine the number of females born in New York in 1953 as compared with the total number of females born in New York in the past 20 years.

```

|IDENTIFICATION DIVISION.
|PROGRAM-ID. TABLES.
|ENVIRONMENT DIVISION.
|CONFIGURATION SECTION.
|SOURCE-COMPUTER. IBM-360.
|OBJECT-COMPUTER. IBM-360.
|SPECIAL-NAMES. CONSOLE IS TYPEWRITER.
|INPUT-OUTPUT SECTION.
|FILE-CONTROL.
|   SELECT INFILE ASSIGN TO SYS007-UT-2400-S-INTAPE.
|   SELECT OUTFILE ASSIGN TO SYS012-UR-1403-S-PRTOUT.
|   SELECT INCARDS ASSIGN TO SYS013-UR-2540R-S-ICARDS.
|DATA DIVISION.
|FILE SECTION.
|FD INFILE LABEL RECORDS ARE OMITTED.
|01 TABLE-1 PIC X(28200).
|01 TABLE-2 PIC X(1800).
|FD OUTFILE LABEL RECORDS ARE OMITTED.
|01 PRTLINE PIC X(133).
|FD INCARDS LABEL RECORDS ARE OMITTED.
|01 CARDS.
|   02 STATE-NAME PIC X(4).
|   02 M-F-CODE PIC 9.
|   02 YEARCODE PIC 9(4).
|   02 FILLER PIC X(71).
|WORKING-STORAGE SECTION.
|01 PRTAREA-20.
|   02 FILLER PIC X VALUE SPACES.
|   02 YEARS-20 PIC 9(4).
|   02 FILLER PIC X(3) VALUE SPACES.
|   02 BIRTHS-20 PIC 9(7).
|   02 FILLER PIC X(3) VALUE SPACES.
|   02 DEATHS-20 PIC 9(7).
|   02 FILLER PIC X(108) VALUE SPACES.
  
```

Figure 51. Sample Table Handling Program (Part 1 of 2)

Table Handling -- Sample Program

```

01  PRTAREA.
    02  FILLER      PIC X.
    02  YEAR       PIC 9(4).
    02  FILLER     PIC X(3) VALUE SPACES.
    02  BIRTHS    PIC 9(5).
    02  FILLER     PIC X(3) VALUE SPACES.
    02  DEATHS    PIC 9(5).
    02  FILLER     PIC X(112) VALUE SPACES.
01  CENSUS-STATISTICS-TABLE.
    02  STATE-TABLE OCCURS 50 TIMES INDEXED BY ST.
    03  STATE-ABBREV PIC X(4).
    03  M-F OCCURS 2 TIMES INDEXED BY SE.
    04  STATISTICS OCCURS 20 TIMES ASCENDING KEY IS YEAR
        INDEXED BY YR.
    05  YEAR      PIC 9(4).
    05  BIRTHS   PIC 9(5).
    05  DEATHS   PIC 9(5).
01  STATISTICS-LAST-20-YRS.
    02  M-F-20 OCCURS 2 TIMES INDEXED BY SE-20.
    03  STATE-20 OCCURS 50 TIMES INDEXED BY ST-20.
    04  YEARS-20 PIC 9(4).
    04  BIRTHS-20 PIC 9(7).
    04  DEATHS-20 PIC 9(7).
PROCEDURE DIVISION.
OPEN-FILES.
    OPEN INPUT INFILE INCARDS OUTPUT OUTFILE.
READ-TABLE.
    READ INFILE INTO CENSUS-STATISTICS-TABLE
        AT END GO TO READ-CARDS.
    READ INFILE INTO STATISTICS-LAST-20-YRS
        AT END GO TO READ-CARDS.
READ-CARDS.
    READ INCARDS
        AT END GO TO EOJ.
DETERMINE-ST.
    SET ST ST-20 TO 1.
    SEARCH STATE-TABLE VARYING ST-20 AT END GO TO ERROR-MSG-1
        WHEN STATE-NAME = STATE-ABBREV (ST) NEXT SENTENCE.
DETERMINE-SE.
    SET SE SE-20 TO M-F-CODE.
DETERMINE-YR.
    SEARCH ALL STATISTICS AT END GO TO ERROR-MSG-2
        WHEN YEAR OF STATISTICS (ST, SE, YR) = YEARCODE
        GO TO WRITE-RECORD.
ERROR-MSG-1.
    DISPLAY "INCORRECT STATE " STATE-NAME UPON TYPEWRITER.
    GO TO READ-CARDS.
ERROR-MSG-2.
    DISPLAY "INCORRECT YEAR " YEARCODE UPON TYPEWRITER.
    GO TO READ-CARDS.
WRITE-RECORD.
    MOVE CORRESPONDING STATISTICS (ST, SE, YR) TO PRTAREA.
    WRITE PRTLINE FROM PRTAREA AFTER ADVANCING 3.
    MOVE CORRESPONDING STATE-20 (SE-20, ST-20) TO PRTAREA-20.
    WRITE PRTLINE FROM PRTAREA-20 AFTER ADVANCING 1.
    GO TO READ-CARDS.
EOJ.
    CLOSE INFILE INCARDS OUTFILE.
    STOP RUN.

```

Figure 51. Sample Program for the Table Handling Feature (Part 2 of 2)

SEGMENTATION FEATURE

The segmentation feature allows the problem programmer to specify object program overlay requirements. The segmentation feature permits segmentation of procedures only. The Procedure Division and Environment Division are considered in determining segmentation requirements for an object program.

ORGANIZATION

Although it is not mandatory, the Procedure Division for a source program is usually written as several consecutive sections, each of which is composed of a series of closely related operations that are designed to perform collectively a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or, to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

FIXED PORTION

The fixed portion is defined as that part of the object program that is logically treated as if it were always in computer storage. This portion of the program is composed of two types of computer storage segments, permanent segments and overlayable fixed segments.

A permanent segment is a segment in the fixed portion that cannot be overlaid by any other part of the program.

An overlayable fixed segment is a segment in the fixed portion which, although logically treated as if it were always in storage, can be overlaid (if necessary) by another segment to optimize storage utilization. However, such a segment, if called for by the program, is always made available in the state it was in when it was last used.

Depending on the availability of storage, the number of permanent segments in the fixed portion can be varied through the use of a special facility called SEGMENT-LIMIT, which is discussed in "Structure of Program Segments."

INDEPENDENT SEGMENTS

An independent segment is defined as that part of the object program which can overlay, and be overlaid by, either an overlayable fixed segment or another independent segment. An independent segment is always considered to be in its initial state each time it is made available to the program.

Segmentation Control and Structure

SEGMENT CLASSIFICATION

Sections that are to be segmented are classified by means of a system of priority numbers. The following criteria should be used:

- Logical requirements: Sections that must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections that are less frequently used are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, depending on logic requirements.
- Frequency of use: Generally, the more frequently a section is referred to, the lower its priority number should be; the less frequently it is referred to, the higher its priority number should be.
- Relationship to other sections: Sections that frequently communicate with one another should be given equal priority numbers. All sections with the same priority number constitute a single program segment.

SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. A reordering of the object module will be necessary if a given segment has its sections scattered throughout the source program. However, the compiler will provide transfers to maintain the logic flow of the source program. The compiler will also insert instructions necessary to load and/or initialize a segment when necessary. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

STRUCTURE OF PROGRAM SEGMENTS

PRIORITY NUMBERS

Section classification is accomplished by means of a system of priority numbers. The priority number is included in the section header.

Format
section-name <u>SECTION</u> [priority-number].

All sections that have the same priority-number constitute a program segment with that priority.

The priority-number must be an integer ranging in value from 0 through 99.

Segments with priority-numbers 0 through 49 belong to the fixed portion of the object program.

Segments with priority-numbers 50 through 99 are independent segments.

Sections in the declaratives portion of the Procedure Division must not contain priority-numbers in their section headers. They are treated as fixed segments with a priority-number of zero.

If the priority-number is omitted from the section header, the priority is assumed to be zero.

When a procedure-name in an independent segment is referred to by a PERFORM statement contained in a segment with a different priority number, the segment referred to is made available in its initial state for each execution of the PERFORM statement.

SEGMENT LIMIT

Ideally, all program segments having priority-numbers ranging from 0 through 49 are treated as permanent segments. However, when insufficient storage is available to contain all permanent segments plus the largest overlayable segment, it becomes necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a means by which he can reduce the number of permanent segments in his program, while these permanent segments still retain the logical properties of fixed portion segments (priority numbers 0 through 49).

Format
[<u>SEGMENT-LIMIT</u> IS priority-number]

The SEGMENT-LIMIT clause is coded in the OBJECT-COMPUTER paragraph.

Priority-number must be an integer that ranges in value from 1 through 49.

When the SEGMENT-LIMIT clause is specified, only those segments having priority-numbers from 0 up to, but not including, the priority number designated as the segment limit are considered as permanent segments of the object program.

Those segments having priority numbers from the segment limit through 49 are considered as overlayable fixed segments.

When the SEGMENT-LIMIT clause is omitted, all segments having priority numbers from 0 through 49 are considered to be permanent segments of the object program.

Segmentation Restrictions

RESTRICTIONS ON PROGRAM FLOW

When segmentation is used, the following restrictions are placed on the ALTER and PERFORM statements and on called programs:

ALTER Statement

1. A GO TO statement in a section whose priority number is 50 or higher must not be referred to by an ALTER statement in a section with a different priority number.
2. A GO TO statement in a section whose priority number is lower than 50 may be referred to by an ALTER statement in any section, even if the GO TO statement to which the ALTER refers is in a segment of the program that has not yet been called for execution.

PERFORM Statement

1. A PERFORM statement that appears in a section whose priority number is lower than the segment limit can have within its range only the following:
 - a. Sections with priority numbers lower than 50.
 - b. Sections wholly contained in a single segment whose priority number is higher than 49.

- However, this compiler allows the PERFORM to have within its range sections with any priority numbers.
2. A PERFORM statement that appears in a section whose priority number is equal to or higher than the segment limit can have within its range only the following:
 - a. Sections with the same priority number as the section containing the PERFORM statement.
 - b. Sections with priority numbers that are lower than the segment limit.

However, this compiler allows the PERFORM to have within its range sections with any priority numbers.

When a procedure-name in a permanent segment is referred to by a PERFORM statement in an independent segment, the independent segment is reinitialized upon exit from the performed paragraphs.

Called Programs

A called program may not be segmented.

SOURCE PROGRAM LIBRARY FACILITY

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures, without recoding them. These entries and procedures are contained in user-created libraries; they are included in a source program by means of a COPY statement.

COPY Statement

The COPY statement permits the user to include prewritten Data Division entries, Environment Division clauses, and Procedure Division procedures in his source program.

Format	
<u>COPY</u>	library-name {SUPPRESS}
<u>{REPLACING}</u>	word-1 <u>BY</u> { word-2 literal-1 identifier-1 }
[word-3 <u>BY</u>	{ word-4 literal-2 identifier-2 }]...].

No other statement or clause may appear in the same entry as the COPY statement, with the exception of the Report Description entry and the File Control Paragraph.

When the library text is copied from the library, compilation is the same as though the text were actually part of the source program.

The COPY statement processing is terminated by the end of the library text.

The text contained in the library must not contain any COPY statements.

Library-name is the name of the library text contained in the user's library. Library-name must follow the rules of formation for program-name. The first eight characters are used as the identifying name.

General Format	
<u>Option 1</u> (within the Configuration Section):	<p><u>SOURCE-COMPUTER</u>. COPY statement. <u>OBJECT-COMPUTER</u>. COPY statement. <u>SPECIAL-NAMES</u>. COPY statement.</p>
<u>Option 2</u> (within the Input-Output Section):	<p><u>FILE-CONTROL</u>. COPY statement. <u>I-O-CONTROL</u>. COPY statement.</p>
<u>Option 3</u> (within the File Control Paragraph):	<p><u>SELECT</u> file-name COPY statement.</p>
<u>Option 4</u> (within the File Section):	<p><u>FD</u> file-name COPY statement. <u>SD</u> sort-file-name COPY statement.</p>
<u>Option 5</u> (within the Report Section):	<p><u>RD</u> report-name COPY statement. <u>CD</u> code-name [OR] CODE mnemonic-name COPY statement.</p>
<u>Option 6</u> (within a File or Sort description entry, or within the Working-Storage Section or the Linkage Section):	<p>01 data-name COPY statement.</p>
<u>Option 7</u> (with a Report Group):	<p>01 [data-name] COPY statement.</p>
<u>Option 8</u> (within the Working-Storage Section or the Linkage Section):	<p>77 data-name COPY statement.</p>
<u>Option 9</u> (within the Working-Storage Section or the Linkage Section):	<p>01 data-name-1 REDEFINES data-name-2 COPY statement. data-name-1 REDEFINES data-name-2 COPY statement.</p>
<u>Option 10</u> (within the Procedure Division):	<p>section-name <u>SECTION</u> [priority-number]. COPY statement. paragraph-name. COPY statement.</p>

The words preceding COPY conform to margin restrictions for COBOL programs. On a given source program card containing the completion of a COPY statement, there must be no information beyond the statement terminating period. The material introduced into the source program by the COPY statement will follow the COPY statement on the listing beginning on the next line. However, the SUPPRESS option may be used to suppress the copy statement.

If the REPLACING option is used, each word specified in the format is replaced by the stipulated word, identifier, or literal which is associated with it in the format.

Word-1, word-2, etc., may be a data-name, procedure-name, condition-name, mnemonic-name, or file-name.

Use of the REPLACING option does not alter the material as it appears in the library.

When options 1, 2, 3, 4, 5, or 10 are written, the words COPY library-name are replaced by the information identified by library-name. This information comprises the sentences or clauses needed to complete the paragraph, sentence, or entry containing the COPY statement.

When options 6, 7, 8, or 9 are written, the entire entry is replaced by the information identified by library-name, except that data-name (if specified) replaces the corresponding data-name from the library.

For example, if the library entry PAYLIB consists of the following Data Division record:

```
01 A.
   02 B          PIC S99.
   02 C          PIC S9(5)V99.
   02 D          PIC S9999 OCCURS 0 TO 52 TIMES
                DEPENDING ON B OF A.
```

the programmer can use the COPY statement in the Data Division of his program as follows:

```
01 PAYROLL COPY PAYLIB.
```

In this program, the library entry is then copied; the resulting entry is treated as if it had been written as follows:

```
01 PAYROLL.
   02 B          PIC S99.
   02 C          PIC S9(5)V99.
   02 D          PIC S9999 OCCURS 0 TO 52 TIMES
                DEPENDING ON B OF A.
```

Note that the data-name A has not been changed in the DEPENDING ON option.

To change some (or all) of the names within the library entry to names he wishes to reference within his program, the programmer can use the REPLACING option:

```
01 PAYROLL COPY PAYLIB REPLACING A BY PAYROLL
   B BY PAY-CODE C BY GROSSPAY.
```

In this program, the library entry is copied; the resulting entry is treated as if it had been written as follows:

```
01 PAYROLL.
   02 PAY-CODE   PIC S99.
   02 GROSS-PAY PIC S9(5)V99.
   02 D         PIC S9999 OCCURS 0 TO 52 TIMES
                DEPENDING ON PAY-CODE OF PAYROLL.
```

The changes shown are made only for this program. The entry as it appears in the library remains unchanged.

EXTENDED SOURCE PROGRAM LIBRARY FACILITY

A complete program may be included as an entry in the user's library and may be used as the basis of compilation. Input to the compiler is a BASIS card, followed by any number of INSERT and/or DELETE cards, followed by any number of debugging packets, if desired. Debug packets must be inserted (using the INSERT card) at the end of the BASIS member (see "Debugging Language").

BASIS Card

Format
<u>BASIS</u> library-name

The word BASIS followed by library-name may appear anywhere within columns 1 through 72 on the card. There must be no other text on the card.

Library-name must follow the rules of formation for program-name. It is the name by which the library entry is known to the control program. The first eight characters are used as the identifying name.

If the INSERT or DELETE cards follow the BASIS card, the library entry is modified prior to being processed by the compiler. Use of INSERT or DELETE cards does not alter the material in the library.

INSERT Card

Format
<u>INSERT</u> sequence-number-field

DELETE Card

Format
<u>DELETE</u> sequence-number-field

The word INSERT or DELETE, followed by a space, followed by sequence-number-field may appear anywhere within columns 1 through 72 on the card. There must be no other text on the card.

Each number in sequence-number-field must refer to a sequence number of the basic library entry. The sequence-number is the six-digit number the programmer assigns in columns 1 through 6 of the COBOL Coding Form.

The numbers in the sequence-number-field must be in ascending numerical order from the first INSERT/DELETE card to the last INSERT/DELETE card in the program.

The sequence-number-field of an INSERT card must be a single number (e.g., 200310). At least one new source program card must follow the INSERT card for insertion after the card specified by the sequence-number-field.

The entries comprising sequence-number-field of a DELETE card must be numbers or ranges of numbers. Each entry must be separated from the preceding entry by a comma followed by a space. Ranges of numbers are indicated by separating the two bounding numbers of the range by a hyphen. For example:

```
000001-000005, 000010
```

Source program cards may follow a DELETE card, for insertion before the card following the last one deleted.

TRACE/EXHIBIT StatementsDEBUGGING LANGUAGE

The following statements are provided for program debugging. They may appear anywhere in an IBM American National Standard COBOL program after the first paragraph name or in a compile-time debugging packet.

For the TRACE and EXHIBIT statements, the output is written on the system list device (SYSLST).

READY/RESET TRACE Statement

Format	
{ <u>READY</u> }	<u>TRACE</u>
{ <u>RESET</u> }	

After a READY TRACE statement is executed, each time execution of a paragraph or section begins, its compiler-generated card number is displayed.

The execution of a RESET TRACE statement terminates the functions of a previous READY TRACE statement.

EXHIBIT Statement

Format	
<u>EXHIBIT</u>	{ <u>NAMED</u> <u>CHANGED</u> <u>NAMED</u> <u>CHANGED</u> }
	{ identifier-1 nonnumeric-literal-1 }
[identifier-2 nonnumeric-literal-2]	...

The execution of an EXHIBIT statement causes a formatted display of the identifiers (or nonnumeric literals) listed in the statement.

Identifiers listed in the statement cannot be any special register except TALLY.

Nonnumeric-literals listed in the statement are followed by a space when displayed.

The display of the operands is continued as described for the DISPLAY statement. A maximum logical record size of 120 characters is assumed.

EXHIBIT NAMED: Each time an EXHIBIT NAMED statement is executed, there is a formatted display of each identifier listed and its value. Since both the identifying name and the value of the identifier are displayed, a fixed columnar format is unnecessary. If the list of operands includes nonnumeric-literals, they are displayed as remarks each time the statement is executed.

The format of the output for each identifier listed in the EXHIBIT NAMED statement is:

```

original identifying name, including qualifiers if written (no more
than 120 characters in length)
space
equal sign
space
value of identifier (no more than 256 bytes in length)
space

```

EXHIBIT CHANGED NAMED: Each time an EXHIBIT CHANGED NAMED statement is executed, there is a display of each identifier listed and its value only if the value has changed since the previous time the statement was executed. The initial time such a statement is executed, all values are considered changed and are displayed. If the list of operands includes nonnumeric-literals, they are displayed as remarks each time the statement is executed.

Since both the identifying name and the value of each identifier is displayed, a fixed columnar format is unnecessary. If some of the identifiers have not changed in value, no space is reserved for them. If none of the identifiers have changed in value, no blank line(s) will be printed.

The format of the output for each identifier listed in the EXHIBIT CHANGED NAMED statement is:

```

original identifying name, including qualifiers if written (no more
than 120 characters in length)
space
equal sign
space
value of identifier (no more than 256 bytes in length)
space

```

EXHIBIT CHANGED: Each time an EXHIBIT CHANGED statement is executed, there is a display of the current value of each identifier listed only if the value has changed since the previous time the statement was executed. The initial time the statement is executed, all values are considered changed and are displayed. If the list of operands includes nonnumeric-literals, they are printed as remarks each time the statement is executed.

The format of the output for a specific EXHIBIT CHANGED statement presents each operand in a fixed columnar position. Since the operands are displayed in the order they are listed in the statement, the programmer can easily distinguish each operand. The following considerations apply:

- If there are two or more identifiers as operands, and some, but not all, are changed from the previous execution of the statement, only the changed values are displayed. The positions reserved for a given operand are blank when the value of the operand has not changed.

ON Statement

- If none of the operands have changed in value from the previous execution of the statement, a blank line(s) will be printed.
- Variable length identifiers are not permitted as operands.
- The storage reserved for any operand cannot exceed 256 bytes.

Note: The combined total length of all operands for all EXHIBIT CHANGED NAMED plus all EXHIBIT CHANGED statements in one program cannot exceed 32,767 bytes.

If two distinct EXHIBIT CHANGED NAMED or two EXHIBIT CHANGED statements appear in one program, each specifying the same identifiers, the changes in value of those identifiers are associated with each of the two separate statements. Depending on the path of program flow, the values of the identifier saved for comparison may differ for each of the two statements.

ON (Count-conditional) Statement

The ON statement allows the programmer to specify when the statements it contains are to be executed.

```
Format 1
-----
ON integer-1 [AND EVERY integer-2] [UNTIL integer-3]
  {imperative-statement ...} {ELSE} {statement ...}
  {NEXT SENTENCE}           {OTHERWISE} {NEXT SENTENCE}
```

```
Format 2 (Version 3)
-----
ON {integer-1} {AND EVERY} {integer-2} ]
   {identifier-1}          {identifier-2}
[UNTIL {integer-3} ] {imperative-statement}
      {identifier-3}   {NEXT SENTENCE}
      {ELSE}          {statement ...}
      {OTHERWISE}     {NEXT SENTENCE}
```

All integers specified in the ON statement must be positive and no greater than 16,777,215.

The phrase ELSE/OTHERWISE NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

Program Product Information -- Version 3

All identifiers must be fixed-point numeric items described as integers. Their values must be positive and no greater than 16,777,215.

At object time each identifier must be initialized to a positive value before the first execution of the ON statement. Between executions of the ON statement, the values contained in the identifiers may be modified. The programmer's manipulation of these values in no way affects the compiler-generated counters associated with the ON statement.

In the following discussion, each reference to integer-1 applies equally to identifier-1. Similarly, each reference to integer-2 applies to identifier-2, and each reference to integer-3 applies to identifier-3.

In all versions of the compiler, the ON statement is evaluated and executed as follows:

- Each ON statement has compiler-generated counters and save areas associated with it. The counters are initialized in the object program. Each time the path of program flow reaches the ON statement, the counters are incremented, and the count-condition (integer-1 AND EVERY integer-2 UNTIL integer-3) is tested.
- If the count-condition is satisfied, the imperative-statement (or NEXT SENTENCE) preceding ELSE/OTHERWISE is executed. (Note that an imperative-statement may consist of a series of imperative statements.)
- If the count-condition is not satisfied, the statement(s) (or NEXT SENTENCE) following ELSE/OTHERWISE is executed. If the ELSE/OTHERWISE option does not appear, the next sentence is executed.

The count-condition is evaluated as follows:

- If only integer-1 has been specified, then the count-condition is satisfied only once: when the path of program flow has reached the ON statement integer-1 times -- that is, when the value in the counter equals integer-1.
- When only integer-1 and integer-3 are specified, then the value of integer-2 is assumed to be one, and the count-condition is satisfied when the value in the counter is any value within the range integer-1 through integer-3.
- If only integer-1 and integer-2 are specified, then the count-condition is satisfied each time the value in the counter is equal to $\text{integer-1} + (\text{integer-2} * K)$, where K is any positive integer or zero. No upper limit for the execution of the ON statement is assumed.
- When all three integers are specified, then the count-condition is satisfied as in the last preceding case, except that an upper limit beyond which the count-condition cannot be satisfied is specified. The upper limit is integer-3.

DEBUG Card

COMPILE-TIME DEBUGGING PACKET

Debugging statements for a given paragraph or section in a program may be grouped together into a debugging packet. These statements will be compiled with the source language program and will be executed at object time. Each packet refers to a specified paragraph-name or section-name in the Procedure Division. Compile-time debugging packets are grouped together and are placed immediately following the source program. No reference to procedure-names in debug packets may be made in the body of the program.

DEBUG Card

Each compile time debug packet is headed by the control card DEBUG.

Format
<u>DEBUG</u> location

The word DEBUG followed by location may appear anywhere within columns 1 through 72 on the card. There must be no other text on the card.

The location is the section-name or paragraph-name (qualified, if necessary) indicating the point in the program at which the packet is to be executed. Effectively, the statements in the packet are executed as though they were physically placed in the source program following the section-name or paragraph-name, but preceding the text associated with the procedure. The same location must not be used in more than one DEBUG control card. Location cannot be a paragraph-name within any DEBUG packet.

A debug packet may consist of any procedural statements conforming to the requirements of COBOL. The following considerations apply:

- A PERFORM or ALTER statement in a debug packet may refer to a procedure-name in any debug packet or in the main body of the Procedure Division.
- A GO TO statement in a debug packet may not refer to a procedure-name in another debug packet, but it may refer to a procedure-name in the main body of the Procedure Division.

FORMAT CONTROL OF THE SOURCE PROGRAM LISTING

There are four statements that allow the programmer to control the spacing of the source program listings produced by the COBOL compiler. These statements are: EJECT, SKIP1, SKIP2, and SKIP3. They may be written anywhere in the source program.

EJECT Statement

The EJECT statement instructs the compiler to print the next source statement at the top of the next page.

Format	
1	Area B

<u>EJECT</u>	

The word EJECT may be written anywhere within Area B and must be the only statement on the card. There must be no punctuation.

SKIP1, SKIP2, and SKIP3 Statements

These statements instruct the compiler to skip 1, 2, or 3 lines before printing the next source statement.

Format	
1	Area B

<u>{SKIP1}</u> <u>{SKIP2}</u> <u>{SKIP3}</u>	

SKIP1 tells the compiler to skip 1 line (double spacing).

SKIP2 tells the compiler to skip 2 lines (triple spacing).

SKIP3 tells the compiler to skip 3 lines (quadruple spacing).

SKIP1, SKIP2, or SKIP3 may be written anywhere within Area B and must be the only statement on the card. There must be no punctuation.

Sterling Conventions

STERLING CURRENCY FEATURE AND INTERNATIONAL CONSIDERATIONS

COBOL provides facilities for handling sterling currency items by means of an extension of the PICTURE clause. Additional options and formats, necessitated by the nondecimal nature of sterling and by the conventions by which sterling amounts are represented in punched cards, are also available.

COBOL provides a means to express sterling currency in pounds, shillings, and pence, in that order. There are 20 shillings in a pound, and 12 pence in a shilling. Although sterling amounts are sometimes expressed in shillings and pence only (in which case the number of shillings may exceed 99), within machine systems, shillings will always be expressed as a two-digit field. Pence, when in the form of integers, likewise will be expressed as a two-digit field. However, provision must be made for pence to be expressed as decimal fractions as well, as in the form 17s.10.237d.

The IBM method for representing sterling amounts in punched cards uses two columns for shillings and one for pence. Tenpence (10d.) is represented by an '11' punch and elevenpence (11d.) by a '12' punch. The British Standards Institution (B.S.I.) representation uses single columns for both shillings and pence. The B.S.I. representation for shillings consists of a '12' punch for ten shillings and the alphabetic punches A through I for 11 through 19 shillings, respectively.

Note: The B.S.I. representation for shillings precludes the use of more than 19 shillings in a sterling expression; therefore, 22/10 (that is, 22 shillings 10 pence) must be expanded by the user to 1/2/10. Similarly, the guinea -- 21 shillings -- or any multiple thereof, must be expanded to pounds and shillings.

The indicated representations may be used separately or in combination, resulting in four possible conventions.

1. IBM shillings and IBM pence
2. IBM shillings and B.S.I. pence
3. B.S.I. shillings and IBM pence
4. B.S.I. shillings and B.S.I. pence

Any of these conventions may be associated with any number of digits (or none) in the pound field and any number of decimal places (or none) in the pence field. In addition, sign representation may be present as an overpunch in one of several allowable positions in the amount, or may be separately entered from another field.

Two formats are provided in the PICTURE clause for the representation of sterling amounts: sterling report format (used for editing) and sterling nonreport format (used for arithmetic).

In the formats that follow, n stands for a positive integer other than zero. This integer enclosed in parentheses and following the symbols 9, B, etc., indicates the number of consecutive occurrences of the preceding symbol. For example, 9(6) and 999999 are equivalent. The PICTURE characters used to describe sterling items are:

6 7 8 9 C D * , / B Z V . £ : s d CR DB + -

(The character £ is the sterling equivalent of the character \$.)

Note: The lower-case letters "s" and "d" are represented by an 11-0-2 punch and a 12-0-4 punch, respectively.

STERLING NONREPORT

The format of the PICTURE clause for a sterling nonreport data item is:

Format	
$\left. \begin{array}{l} \text{PICTURE} \\ \text{PIC} \end{array} \right\}$	IS 9[(n)]D[8]8D $\left. \begin{array}{l} \{6[6]\} \\ \{7[7]\} \end{array} \right\}$ [[V]9[(n)]] [USAGE IS] DISPLAY-ST

Note: For a sterling nonreport picture to be valid, it must contain a pound field, a shilling field, and a pence field.

The representation for pounds is 9[(n)]D where:

1. The character 9 indicates that a character position will always contain a numeric character, and may extend to n positions.
2. The character D indicates the position of an assumed pound separator.

The representation for shillings is [8]8D where:

1. The characters [8]8 indicate the position of the shilling field and the convention by which shillings are represented in punched cards. 88 indicates IBM shilling representation occupying a two-column field. 8 indicates B.S.I. single-column shilling representation.
2. The character D indicates the position of an assumed shilling separator.

The representation for pence is:

$$\left. \begin{array}{l} \{6[6]\} \\ \{7[7]\} \end{array} \right\} \quad [[V]9[(n)]]$$

1. The character 6 indicates IBM single-column pence representation wherein 10d. is represented by an '11' punch and 11d. by a '12' punch. The characters 66 indicate two-column representation of pence, usually from some external medium other than punched cards.

Sterling Sign Representation

2. The character 7 indicates B.S.I. single-column pence representation wherein 10d. is represented by a '12' punch and 11d. by an '11' punch. The characters 77 indicate two-column representation of pence. Consequently, 66 and 77 serve the same purpose and are interchangeable.
3. The character V indicates the position of an assumed decimal point in the pence field. Its properties and use are identical with that of V in dollar amounts. Decimal positions in the pence field may extend to n positions.
4. The character 9 indicates that a character position will always contain a numeric character, and may extend to n positions.

Example: Assume that a sterling currency data item used in arithmetic expressions is to be represented in IBM shillings and IBM pence, and that this data item will never exceed 99/19s/11d. Its picture should be:

```
PICTURE 9(2)D88D6 DISPLAY-ST.
```

The VALUE clause must not be specified for sterling nonreport items.

Sterling Sign Representation

Signs for sterling amounts may be entered as overpunches in one of several allowable positions of the amount. A sign is indicated by an embedded S in the nonreport PICTURE immediately to the left of the position containing the overpunch. Allowable overpunch positions are the high-order and low-order positions of the pound field, the high-order shilling digit in two-column shilling representation, the low-order pence digit in two-column pence representation, or the least significant decimal position of pence.

The following are examples of sterling currency nonreport data items showing sign representation in each of the allowable positions:

```
PICTURE S99D88D6V9(3) DISPLAY-ST
```

```
PICTURE 9S9D88D6V9(3) DISPLAY-ST
```

```
PICTURE 9(2)DS88D6V9(3) DISPLAY-ST
```

```
PICTURE 9(2)D88D6S6V9(3) DISPLAY-ST
```

```
PICTURE 9(2)D88D6V99S9 DISPLAY-ST
```


STERLING REPORT

The sterling currency report data is composed of four portions: pounds, shillings, pence, and pence decimal fractions.

FORMAT	
<u>PICTURE</u>	} IS
<u>PIC</u>	
[pound-report-string]	[pound-separator-string] delimiter
shilling-report-string	[shilling-separator-string] delimiter
pence-report-string	[pence-separator-string][sign-string]
<u>[USAGE IS] DISPLAY-ST</u>	

Pound-Report-String - This string is optional. It is subject to the same rules as other numeric edited items, with the following exceptions:

- The allowable characters are: £ (pound symbol) 9 Z * + - 0 (zero) B , (comma).
- The total number of digits in the pound-report-string plus the fractional-pence field cannot exceed 15. (That is, if there are 11 digits in the pound-report-string, there cannot be more than 4 digits in the fractional-pence-field.)
- The character £ is the sterling equivalent of \$.
- Termination is controlled by the pound-separator-string.

Pound-Separator-String - This string is optional. It may include one character, or any combination of the following characters:

B : / . (period or decimal point)

Editing of the separator characters is dependent upon the use of C or D as delimiters.

The Delimiter Characters - The delimiter characters C and D are required. They primarily serve to indicate the end of the pounds and shillings portions of the picture. In addition, they serve to indicate the type of editing to be applied to separator characters to the right of the low-order digit (of the pounds and shillings integer portions of the item).

The delimiter character D indicates that separator character(s) to the right of the low-order digit position (of the field delimited) are always to appear; that is, no editing is performed on the separator character(s).

The delimiter character C indicates that if the low-order digit position (of the field delimited) is represented by other than the edit character 9, editing continues through the separator character(s).

The delimiter characters C and D are used for editing purposes only. They do not take up space in the printed result.

Sterling Report Format

The following examples show the editing performed when a value of zero is moved to a sterling report item.

```
**/CZ9s/D99d
```

would result in

```
***b0s/00d
```

whereas, if the picture were

```
**/DZ9s/D99d
```

the result would be

```
**/b0s/00d
```

The delimiter C is equivalent to D when the low-order digit position is represented by a 9. That is, the following two pictures are equivalent:

```
ZZ9/DZ9/D99
```

```
ZZ9/CZ9/C99
```

The delimiters used for the pounds and shillings portion of the picture need not be the same.

Note: Although the pound-report-string and the pound-separator-string are optional, a delimiter character (either C or D) must be present; thus, when programming for shillings and pence only, the PICTURE clause must begin PICTURE IS C (or D).

Shilling-Report-String - This is a required two-character field. It is made up of the following characters:

```
9 8 X *
```

The valid combinations of these characters are:

```
99 Z9 ZZ Z8 *9 ** *8
```

The 8 is an edit character and is treated as a 9. However, if the digits to the left of the edit character 8 are zeros, the 8 is treated as the character that precedes it (either Z or *).

Shilling-Separator-String - This string is optional. It may include one character, or any combination of the following characters:

```
B : / s . (period or decimal point)
```

Editing of the shilling-separator characters is dependent upon the use of C or D as delimiters.

Pence-Report-String - This field is made up of two parts: a required whole-pence field, and an optional fractional-pence field.

The required whole-pence field is a two-character field, made up of the following symbols:

```
9 8 Z *
```

Valid combinations of these characters are:

```
99 Z9 ZZ Z8 *9 ** *8
```

The function of the editing character 8 is the same as in the shilling-report-string.

The optional fractional-pence field is indicated by a decimal point followed by one or more 9's. It is used to specify fractional pence in decimal form.

The total number of digits in the fractional-pence field plus the pound-report-string cannot exceed 15.

Pence-Separator-String - This string is optional and may consist of one or both of the following characters:

d . (period or decimal point)

If both characters are used, they must be used in the order shown above.

Sign-Field - This field is optional and may consist of:

- optionally, one or more blanks (B), followed by
- one of the following one- or two-character combinations:

+ - CR DB

Sterling Report editing applications are shown in Figure 52.

Picture	Numeric Value (in pence)	Sterling Equivalent £ s d	Printed Result
££/D99s/D99d	3068	12 15 08	£12/15s/08d
££/D99s/D99d	0668	2 15 08	£2/15s/08d
££/D99s/D99d	0188	0 15 08	/15s/08d
££:C99s:D99d	0188	0 15 08	15s/08d
ZZZ/DZZs/DZZd	0000	0 00 00	/ s/ d
ZZZ/CZZs/DZZd	0000	0 00 00	s/ d
BD99sBD99.9d	080.5	0 06 08.5	06s 08.5d
***C**D/C**99d	1040.12	4 06 08.12	**4/*6s/*8.12d
***:C**s:C**99d	080.12	0 06 08.12	***6s:*8.12d
***D**s/D**99d	00001.23	0 00 01.23	***/**s/*1.23d
££/D*9s/D**99d	00961.23	4 00 01.23	£4/*0s/*1.23d
£**/D*9s/D**99d	00961.23	4 00 01.23	£*4/*0s/*1.23d
£**/D*9s/D**99d	00001.23	0 00 01.23	£**/*0s/*1.23d
££/D99s/D99dCR	-3068	12 15 08	£12/15s/08dCR

Figure 52. Sterling Currency Editing Applications

A sterling report PICTURE may have a BLANK WHEN ZERO clause associated with it specifying that the item described is filled with spaces whenever the value of the item is zero.

If the VALUE clause is specified for a Sterling Report item, the literal must be alphanumeric. The VALUE clause is treated exactly as it is specified, with no editing performed.

The maximum length of a sterling report item is 127 characters.

International Currency Considerations

PROCEDURE DIVISION CONSIDERATIONS

The MOVE, DISPLAY, ACCEPT, EXAMINE, and TRANSFORM statements, arithmetic statements, and relation tests may be written containing identifiers that represent sterling items.

If a nonsterling value is moved into a Sterling report item, the compiler treats the value as pence, and converts it to pounds/shillings/pence.

Sterling items are not considered to be integers and should not be used where an integer is required.

INTERNATIONAL CONSIDERATIONS

1. The functions of the period and the comma may be exchanged in the PICTURE character-string and in numeric literals by writing the clause DECIMAL-POINT IS COMMA in the SPECIAL-NAMES paragraph of the Environment Division.
2. The PICTURE of report items may terminate with the currency symbol in cases where the graphic \$ is supplanted by a particular national currency symbol, through use of the CURRENCY SIGN IS literal clause in the SPECIAL-NAMES paragraph of the Environment Division.

- APPENDIXES

- A: Intermediate Results

- B: Sample Programs

- C: American National Standard COBOL Formats and Reserved Words

- D: File-Processing Techniques and Applicable Statements and Clauses

- E: ASCII Considerations (Version 3)

- F: Symbolic Debugging Feature (Version 3)

- G: Combined Function Card Processing (Version 3)

- IBM American National Standard COBOL Glossary

This appendix discusses the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results. The following abbreviations are used:

i--number of integer places carried for an intermediate result

d--number of decimal places carried for an intermediate result

dmax--maximum number of decimal places defined for any operand (except for floating-point operands, exponents, or divisors) in a particular statement

op1--first operand in a generated arithmetic statement

op2--second operand in a generated arithmetic statement

d1,d2--number of decimal places defined for op1 or op2, respectively

ir--intermediate result field obtained from the execution of a generated arithmetic statement or operation. ir1, ir2, etc., represent successive intermediate results. These intermediate results are generated either in registers or in storage locations. Successive intermediate results may have the same location.

In the case of an arithmetic statement containing only a single pair of operands, no intermediate results are generated, except when the TRUNC option is specified for COMPUTATIONAL items. Intermediate results are possible in the following cases:

1. In an ADD or SUBTRACT statement containing multiple operands immediately following the verb.
2. In a COMPUTE statement for a series of arithmetic operations.
3. In arithmetic expressions contained in IF or PERFORM statements.

In such cases, the compiler treats the statement as a succession of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

is replaced by

** F	BY G	yielding ir1
MULTIPLY B	BY C	yielding ir2
DIVIDE E	INTO D	yielding ir3
ADD A	TO ir2	yielding ir4
SUBTRACT ir3	FROM ir4	yielding ir5
ADD ir5	TO ir1	yielding Y

COMPILER CALCULATION OF INTERMEDIATE RESULTS

The number of integer places in an ir is calculated as follows:

- The compiler first determines the maximum value the ir can contain by performing the statement in which the ir occurs.

1. If an operand in this statement is a data-name, the value used for the data-name is equal to the numerical value of the PICTURE for data-name (e.g., PICTURE 9V99 has value 9.99).
 2. If an operand is a literal, the literal's actual value is used.
 3. If an operand is an intermediate result, the value determined for the intermediate result in a previous arithmetic operation is used.
 4. If the operation is division:
 - a. If op2 is a data-name, the value used for op2 is the minimum nonzero value of the digit in the PICTURE for the data-name (e.g., PICTURE 9V99 has the value 0.01).
 - b. If op2 is an intermediate result, the intermediate result is treated as though it had a PICTURE, and the minimum nonzero value of the digits in this PICTURE is used.
- When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.
 - The number of decimal places contained in an ir is calculated as:

<u>Operation</u>	<u>Decimal Places</u>
+ or -	d1 or d2, whichever is greater
*	d1 + d2
/	d1 - d2 or dmax, whichever is greater
**	dmax if op2 is nonintegral or a data-name; d1 * op2 if op2 is an integral literal

Figure 53 indicates the action of the compiler when handling intermediate results.

Value of <u>i + d</u>	Value of <u>d</u>	Value of <u>i + dmax</u>	Action Taken
<30 =30	Any value	Any value	<u>i</u> integer and <u>d</u> decimal places are carried for <u>ir</u>
>30	<dmax	Any value	30 - <u>d</u> integer and <u>d</u> decimal places are carried for <u>ir</u>
	=dmax		
	>dmax	<30 =30 >30	<u>i</u> integer and 30 - <u>i</u> decimal places are carried for <u>ir</u> 30 - <u>dmax</u> integer and <u>dmax</u> decimal places are carried for <u>ir</u>

Figure 53. Compiler Action on Intermediate Results

The three programs in this appendix illustrate several methods of accessing mass storage files. The three programs are:

1. CREATION OF A DIRECT FILE



CREATION OF A DIRECT FILE

This program creates a file with direct organization through use of an ACTUAL KEY -- using actual track addressing. The ACTUAL KEY has two components:

1. A track identifier, which is the actual track address, calculated through a simple remainder randomizing technique. A field from the input record (CD-ITEM-CODE) is converted to a track address (TRACK1). CD-ITEM-CODE is divided by 19; the remainder is placed in TRACK1. This gives a valid track address ranging in value from 0 to 18. The program assumes that the file must begin on cylinder 102; therefore, a displacement value of 1020 is added to assure that the CC field will contain the value of decimal 102.
2. A unique record identifier, in this case TRACK-NAME, that contains the CD-ITEM-NAME of the input record.

The UPDATING program in the introduction updates the file that this program creates.

IDENTIFICATION DIVISION.

PROGRAM-ID. CREATEDF.

REMARKS. ILLUSTRATE CREATION OF A DIRECT FILE. THE FILE MAY BE USED AS THE MASTER FILE FOR THE SAMPLE PROGRAM ILLUSTRATED IN THE INTRODUCTION TO THIS MANUAL.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F50.

OBJECT-COMPUTER. IBM-360-F50.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT DA-FILE ASSIGN TO SYS015-DA-2311-A-MASTER

ACCESS IS RANDOM

ACTUAL KEY IS FILEKEY.

SELECT CARD-FILE ASSIGN TO SYS007-UR-2540R-S.

DATA DIVISION.

FILE SECTION.

FD DA-FILE

DATA RECORD IS DISK

LABEL RECORDS ARE STANDARD.

01 DISK.

02 DISK-ITEM-CODE PICTURE X(3).

02 DISK-ITEM-NAME PICTURE X(29).

02 DISK-STOCK-ON-HAND PICTURE S9(6) USAGE COMP SYNC.

02 DISK-UNIT-PRICE PICTURE S999V99 USAGE COMP SYNC.

02 DISK-STOCK-VALUE PICTURE S9(9)V99 USAGE COMP SYNC.

02 DISK-ORDER-POINT PICTURE S9(3) USAGE COMP SYNC.

FD CARD-FILE

LABEL RECORDS ARE OMITTED

DATA RECORD IS CARDS.

01 CARDS.

02 CD-ITEM-CODE PICTURE X(3).

02 CD-ITEM-NAME PICTURE X(29).

02 CD-STOCK-ON-HAND PICTURE S9(6).

02 CD-UNIT-PRICE PICTURE S999V99.

02 CD-STOCK-VALUE PICTURE S9(9)V99.

02 CD-ORDER-POINT PICTURE S9(3).

02 FILLER PICTURE X(23).

```

WORKING-STORAGE SECTION.
77 SAVE PICTURE S9(5) USAGE COMP SYNC.
77 QUOTIENT PICTURE S9(4) USAGE COMP SYNC.
01 TRACK1 PICTURE 9999.
01 TRACK2 REDEFINES TRACK1.
  02 CYL PICTURE 999.
  02 HEAD PICTURE 9.
01 KEY-1.
  02 M PICTURE S999 COMP SYNC VALUE ZEROES.
  02 BB PICTURE S9 COMP SYNC VALUE ZERO.
  02 CC PICTURE S999 COMP SYNC.
  02 HH PICTURE S999 COMP SYNC.
  02 R PICTURE X VALUE LOW-VALUE.
  02 TRACK-NAME PICTURE X(29).
01 THE-KEY REDEFINES KEY-1.
  02 FILLER PICTURE X.
  02 FILEKEY PICTURE X(37).

PROCEDURE DIVISION.
BEGIN.
  OPEN INPUT CARD-FILE.
  OPEN OUTPUT DA-FILE.
PARA-1.
  READ CARD-FILE AT END GO TO END-JOB.
  MOVE CD-ITEM-CODE TO SAVE.
  DIVIDE 19 INTO SAVE GIVING QUOTIENT
    REMAINDER TRACK1.
  ADD 1020 TO TRACK1.
  MOVE CD-ITEM-NAME TO TRACK-NAME.
  MOVE CYL TO CC.
  MOVE HEAD TO HH.
  MOVE CD-ITEM-CODE TO DISK-ITEM-CODE.
  MOVE CD-ITEM-NAME TO DISK-ITEM-NAME.
  MOVE CD-STOCK-ON-HAND TO DISK-STOCK-ON-HAND.
  MOVE CD-UNIT-PRICE TO DISK-UNIT-PRICE.
  MOVE CD-STOCK-VALUE TO DISK-STOCK-VALUE.
  MOVE CD-ORDER-POINT TO DISK-ORDER-POINT.
WR.
  WRITE DISK INVALID KEY GO TO ERROR-ROUTINE.
  GO TO PARA-1.
END-JOB.
  CLOSE CARD-FILE DA-FILE.
  DISPLAY "END OF JOB".
  STOP RUN.
ERROR-ROUTINE.
  DISPLAY "UNABLE TO WRITE RECORD".
  DISPLAY TRACK-NAME.
  GO TO PARA-1.

```

This program creates an indexed file. These records are presented in ascending sequence by RECORD KEY. The APPLY clause builds the master index.

IDENTIFICATION DIVISION.

PROGRAM-ID. CREATEIS.

REMARKS. ILLUSTRATE CREATION OF INDEXED SEQUENTIAL FILE.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F50.

OBJECT-COMPUTER. IBM-360-F50.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT IS-FILE ASSIGN TO SYS015-DA-2311-I-MASTER

ACCESS IS SEQUENTIAL

RECORD KEY IS REC-ID.

SELECT CARD-FILE ASSIGN TO SYS007-UR-2540R-S

RESERVE 1 ALTERNATE AREA.

I-O-CONTROL. APPLY MASTER-INDEX TO 2311 ON IS-FILE.

DATA DIVISION.

FILE SECTION.

FD IS-FILE BLOCK CONTAINS 5 RECORDS

RECORDING MODE IS F

LABEL RECORDS ARE STANDARD.

01 DISK.

02 DELETE-CODE PICTURE X.

02 REC-ID PICTURE 9(10).

02 DISK-FLD1 PICTURE X(10).

02 DISK-NAME PICTURE X(20).

02 DISK-BAL PICTURE 99999V99.

02 FILLER PICTURE X(52).

FD CARD-FILE RECORDING MODE IS F

LABEL RECORDS ARE OMITTED.

01 CARDS.

02 KEY-ID PICTURE 9(10).

02 CD-NAME PICTURE X(20).

02 CD-BAL PICTURE 99999V99.

02 FILLER PICTURE X(43).

PROCEDURE DIVISION.

BEGIN.

OPEN INPUT CARD-FILE.

OPEN OUTPUT IS-FILE.

PARA-1.

READ CARD-FILE AT END GO TO END-JOB.

MOVE LOW-VALUE TO DELETE-CODE.

MOVE KEY-ID TO REC-ID.

MOVE CD-NAME TO DISK-NAME.

MOVE CD-BAL TO DISK-BAL.

WRITE DISK INVALID KEY GO TO ERR.

GO TO PARA-1.

ERR.

DISPLAY "DUPLICATE OR SEQ-ERR" UPON CONSOLE.

DISPLAY KEY-ID UPON CONSOLE.

GO TO PARA-1.

END-JOB.

CLOSE CARD-FILE IS-FILE.

DISPLAY "END OF JOB" UPON CONSOLE.

STOP RUN.

RANDOM RETRIEVAL AND UPDATING OF AN INDEXED FILE

This program randomly updates an existing indexed file. The READ IS-FILE statement causes a search of indexes for an equal compare between the NOMINAL KEY obtained from the input record and the RECORD KEY of the I-O file. If an equal compare occurs, the record is updated, and the details of this update are printed. If a matching record is not found, the invalid key branch is taken.

IDENTIFICATION DIVISION.

PROGRAM-ID. RANDOMIS.

REMARKS. ILLUSTRATE RANDOM RETRIEVAL FROM IS-FILE.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-360-F50.

OBJECT-COMPUTER. IBM-360-F50.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT IS-FILE ASSIGN TO SYS015-DA-2311-I-MASTER
ACCESS IS RANDOM

NOMINAL KEY IS KEY-ID

RECORD KEY IS REC-ID.

SELECT CARD-FILE ASSIGN TO SYS007-UR-2540R-S

RESERVE 1 ALTERNATE AREA.

SELECT PRINT-FILE ASSIGN TO SYS008-UT-2400-S-PROUT

RESERVE NO ALTERNATE AREAS.

I-O-CONTROL.

APPLY MASTER-INDEX TO 2311 ON IS-FILE.

RERUN ON SYS002-UT-2400-S-CKPT EVERY 10000 RECORDS
OF IS-FILE.

DATA DIVISION.

FILE SECTION.

FD IS-FILE

BLOCK CONTAINS 5 RECORDS

RECORD CONTAINS 100 CHARACTERS

LABEL RECORDS ARE STANDARD

RECORDING MODE IS F

DATA RECORD IS DISK.

01 DISK.

02 DELETE-CODE PICTURE X.

02 REC-ID PICTURE 9(10).

02 DISK-FLD1 PICTURE X(10).

02 DISK-NAME PICTURE X(20).

02 DISK-BAL PICTURE 99999V99.

02 FILLER PICTURE X(52).

FD CARD-FILE

RECORDING MODE IS F

LABEL RECORDS ARE OMITTED

DATA RECORD IS CARDS.

01 CARDS.

02 KEY-IDA PICTURE 9(10).

02 CD-NAME PICTURE X(20).

02 CD-AMT PICTURE 99999V99.

02 FILLER PICTURE X(43).

FD PRINT-FILE

RECORDING MODE IS F

LABEL RECORDS ARE STANDARD

DATA RECORD IS PRINTER.

```

01  PRINTER.
    02  FORMSC          PICTURE X.
    02  PRINT-ID       PICTURE X(10).
    02  FILLER         PICTURE X(10).
    02  PRINT-NAME     PICTURE X(20).
    02  FILLER         PICTURE X(10).
    02  PRINT-BAL      PICTURE $ZZZ,999.99-.
    02  FILLER         PICTURE X(10).
    02  PRINT-AMT      PICTURE $ZZZ,ZZZ.99-.
    02  FILLER         PICTURE X(10).
    02  PRINT-NEW-BAL  PICTURE $ZZZ,ZZZ.99-.
WORKING-STORAGE SECTION.
77  KEY-ID            PICTURE 9(10).
PROCEDURE DIVISION.
BEGIN.
    OPEN INPUT CARD-FILE.
    OPEN OUTPUT PRINT-FILE.
    OPEN I-O IS-FILE.
PARA-1.
    MOVE SPACES TO PRINTER.
    READ CARD-FILE AT END GO TO END-JOB.
    MOVE KEY-IDA TO KEY-ID.
    READ IS-FILE INVALID KEY GO TO NO-RECORD.
    MOVE REC-ID TO PRINT-ID.
    MOVE DISK-NAME TO PRINT-NAME.
    MOVE DISK-BAL TO PRINT-BAL.
    MOVE CD-AMT TO PRINT-AMT.
    ADD CD-AMT TO DISK-BAL.
    MOVE DISK-BAL TO PRINT-NEW-BAL.
    REWRITE DISK INVALID KEY GOTO NO-RECORD.
    WRITE PRINTER AFTER POSITIONING 2 LINES.
    GO TO PARA-1.
NO-RECORD.
    DISPLAY 'NO RECORD FOUND' UPON CONSOLE.
    DISPLAY KEY-ID UPON CONSOLE.
    GO TO PARA-1.
END-JOB.
    CLOSE CARD-FILE PRINT-FILE IS-FILE.
    DISPLAY 'END OF JOB' UPON CONSOLE.
    STOP RUN.

```

APPENDIX C: AMERICAN NATIONAL STANDARD COBOL FORMAT SUMMARY AND RESERVED WORDS

The Formats and Reserved Words in this appendix have been printed in a specially reduced size with pages numbered in sequence to make up a pocket-sized reference booklet for use while coding IBM American National Standard COBOL programs. Although most readers may prefer to retain this reference material within the manual, the booklet can be prepared as follows:

- Cut along trim lines.
- Place sheets so that page numbers at lower right-hand corner are in ascending order in odd-number progression (i.e., 1, 3, 5, etc.); lower left-hand page numbers will then be in descending order in even-number progression (i.e., 16, 14, 12, etc.).
- Fold trimmed sheets after collating.
- Staple along fold if desired.
- Punch for six-hole binder.

FOLD
TRIM HERE

TRIM HERE

Symbols Allowed in the PICTURE Clause

Symbol	Meaning
A	Alphabetic character or space
B	Space insertion character
E	External floating-point exponent
F	Decimal scaling position (not counted in size of data item)
S	Operational sign (not counted in size of data item)
V	Assumed decimal point (not counted in size of data item)
X	Alphanumeric character (any from the EBCDIC set)
Z	Zero suppression character
9	Numeric character
0	Zero insertion character
.	Common insertion character
+	Decimal point or period editing control character
-	Plus sign insertion editing control character
CR	Minus sign editing control character
DB	Credit editing control characters
*	Debit editing control characters
^	Check protect insertion character
~	Currency sign insertion character

IBM Reference Data

DOS

IBM Full American National Standard COBOL

Appendix C: IBM Full American National Standard COBOL Format Summary and Reserved Words

The general format of a COBOL program is illustrated in these format summaries. Included within the general format is the specific format for each valid COBOL statement. All clauses are shown as though they were required by the COBOL source program, although within a given context many are optional. Repetition of clauses has not been indicated. Several formats are included under special headings, which are different from, or additions to, the general format. Under these special headings are included formats peculiar to the following COBOL features: Sort, Report Writer, Table Handling, Segmentation, Source Program Library Facility, Debugging Language, Format Control of the Source Program Listing, and Sterling Currency. Each of these features is explained within a special chapter of this publication — *IBM DOS Full American National Standard COBOL*, Order No. GC28-6394-4.

Note: Formats valid only for DOS/VS COBOL are shown in a separate section following the reserved word list.



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza
New York, New York 10017
(International)

Printed in U.S.A.

Extracted from GC28-6394-4
Not orderable separately

TRIM HERE
FOLD

FOLD

TRIM HERE

IDENTIFICATION DIVISION — BASIC FORMATS

{
 IDENTIFICATION DIVISION }
 NO DIVISION }
 PROGRAM-ID program-name.
 AUTHOR [comment-entry] ...
 INSTALLATION [comment-entry] ...
 DATE-WRITTEN [comment-entry] ...
 DATE-COMPILED [comment-entry] ...
 SECURITY [comment-entry] ...
 REMARKS [comment-entry] ...

ENVIRONMENT DIVISION — BASIC FORMATS

ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER computer-name.
 OBJECT-COMPUTER computer-name [MEMORY SIZE integer { WORDS CHARACTERS MODULES }]
 [SEGMENT-LIMIT IS priority-number].
 SPECIAL-NAMES. [function-name-1 IS mnemonic-name] ...
 [function-name-2 IS mnemonic-name]
 [ON STATUS IS condition-name-1 [OFF STATUS IS condition-name-2]] ...
 [OFF STATUS IS condition-name-2 [ON STATUS IS condition-name-1]] ...
 [CURRENCY SIGN IS literal]
 [DECIMAL-POINT IS COMMA].

INPUT-OUTPUT SECTION.

FILE-CONTROL.
 {SELECT [OPTIONAL] file-name
 ASSIGN TO [integer] system-name-1 [system-name-2] ...
 [FOR MULTIPLE {REEL UNIT}]
 RESERVE {NO integer} ALTERNATE [AREA AREAS]
 {FILE-LIMIT IS {data-name-1} THRU {data-name-2}
 FILE-LIMITS ARE {literal-1} THRU {literal-2}
 [{data-name-3} THRU {data-name-4}] ...
 ACCESS MODE IS {SEQUENTIAL RANDOM}
 PROCESSING MODE IS SEQUENTIAL
 ACTUAL KEY IS data-name
 NOMINAL KEY IS data-name
 RECORD KEY IS data-name
 TRACK-AREA IS integer CHARACTERS }

I-O-CONTROL.
 RERUN ON system-name EVERY integer RECORDS OF file-name
 SAME [SORT RECORD] AREA FOR file-name-1 [file-name-2] ...
 MULTIPLE FILE TAPE CONTAINS file-name-1 [POSITION integer-1]
 [file-name-2 [POSITION integer-2]] ...

READ Statement

FORMAT 1
 READ file-name [NEXT] RECORD [INTO identifier]
 [AT END imperative-statement]

FORMAT 2
 READ file-name RECORD [INTO identifier]
 [INVALID KEY imperative-statement]

REWRITE Statement
 REWRITE record-name [FROM identifier]
 [INVALID KEY imperative-statement]

START Statement
 START file-name {KEY IS { EQUAL TO
 GREATER THAN
 NOT LESS THAN
 NOT < }
 [INVALID KEY imperative-statement]

USE Sentence
 USE AFTER STANDARD {EXCEPTION ERROR} PROCEDURE
 ON { file-name-1 [file-name-2] ...
 INPUT
 OUTPUT
 LO
 EXTEND }

WRITE Statement
 WRITE record-name [FROM identifier]
 [INVALID KEY imperative-statement]

MERGE FACILITY FORMATS (DOS/VS COBOL Only)

Environment Division — Input-Output Section

FILE-CONTROL Entry
 FILE-CONTROL.
 {SELECT file-name
 ASSIGN TO system-name-1 [system-name-2] ... }
 I-O-CONTROL Entry
 I-O-CONTROL.
 SAME { SORT SORT-MERGE } AREA FOR file-name-1 [file-name-2] ...
 RECORD

Data Division — Merge File Description Entry

SD merge-file-name
 RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
 DATA {RECORD IS data-name-1 [data-name-2] ...
 RECORDS ARE }
 SORT-OPTION IS data-name-3.

Procedure Division — Merge Statement

MERGE file-name-1
 ON {ASCENDING DESCENDING} KEY data-name-1 [data-name-2] ...
 [ON {ASCENDING DESCENDING} KEY data-name-3 [data-name-4] ...] ...
 USING file-name-2 file-name-3 [file-name-4] ...
 {GIVING file-name-5
 OUTPUT PROCEDURE IS section-name-1 [THRU section-name-2]}

TRIM HERE

TRIM HERE

FOLD

FOLD

TRIM HERE

WORKING-STORAGE SECTION.

77 *data-name-1*

01-49 {*data-name-1*}
 {FILLER}

REDEFINES *data-name-2*

BLANK WHEN ZERO

{JUSTIFIED} RIGHT
 {JUST}

{PICTURE} IS character string
 {PIC}

{SIGN IS} {LEADING} {TRAILING} {SEPARATE CHARACTER} (Version 3)

{SYNCHRONIZED} {LEFT} {RIGHT}
 {SYNC}

{USAGE IS} INDEX
 DISPLAY
 COMPUTATIONAL
 COMP
 COMPUTATIONAL-1
 COMP-1
 COMPUTATIONAL-2
 COMP-2
 COMPUTATIONAL-3
 COMP-3
 COMPUTATIONAL-4 (Version 3)
 COMP-4
 DISPLAY-ST

VALUE IS literal.

88 *condition-name* {VALUE IS
 VALUES ARE} *literal-1* [THRU *literal-2*]
 [*literal-3* [THRU *literal-4*]] . . .

86 *data-name-1* RENAMES *data-name-2* [THRU *data-name-3*].

Note: Formats of the OCCURS Clause are included with Formats for the TABLE HANDLING feature.

LINKAGE SECTION.

77 *data-name-1*

01-49 {*data-name-1*}
 {FILLER}

REDEFINES *data-name-2*

BLANK WHEN ZERO

{JUSTIFIED} RIGHT
 {JUST}

{PICTURE} IS character string
 {PIC}

{SIGN IS} {LEADING} {TRAILING} {SEPARATE CHARACTER} (Version 3)

{SYNCHRONIZED} {LEFT} {RIGHT}
 {SYNC}

{USAGE IS} INDEX
 DISPLAY
 COMPUTATIONAL
 COMP
 COMPUTATIONAL-1
 COMP-1
 COMPUTATIONAL-2
 COMP-2
 COMPUTATIONAL-3
 COMP-3
 COMPUTATIONAL-4 (Version 3)
 COMP-4
 DISPLAY-ST

88 *condition-name* {VALUE IS
 VALUES ARE} *literal-1* [THRU *literal-2*]
 [*literal-3* [THRU *literal-4*]] . . .

86 *data-name-1* RENAMES *data-name-2* [THRU *data-name-3*].

Note: Formats of the OCCURS Clause are included with Formats for the TABLE HANDLING feature.

(xac) PRINT-SWITCH

(ca) PROCEDURE

(ca) PROCEDURE

(ca) PROCEED

(ca) PROCESS

(xa) PROCESSING

(xa) PROGRAM

(ca) PROGRAM-ID

(spn) QUEUE

(ca) QUOTE

(spn) QUOTES

RANDOM

RD

(xac) READY

(spn) RECEIVE

(spn) RECORD

(spn) RECORD-OVERFLOW

(xa) RECORDING

(ca) RECORDS

(ca) REDEFINES

(ca) REFERENCES

(ca) REEL

(ca) RELATIVE

(spn) RELEASE

(spn) RELOAD

(ca) REMAINDER

(ca) REMARKS

(ca) REMOVAL

(spn) RENAMES

(spn) REORG-CRITERIA

(ca) REPLACING

(spn) REPORT

(spn) REPORTING

(spn) REPORTS

(ca) RERUN

(ca) RESERVE

(spn) RESET

(spn) RETURN

(spn) RETURN-CODE

(ca) REVERSED

(xa) REWIND

(ca) REWRITE

RF

RH

RIGHT

ROUNDED

RUN

(ca) SA

(ca) SAME

SD

SEARCH

SECTION

SECURITY

SEEK

(spn) SEGMENT

(ca) SEGMENT-LIMIT

(spn) SELECT

(spn) SEND

(xa) SENTENCE

(ca) SEPARATE

(spn) SEQUENTIAL

(spn) SERVICE

SET

SIGN

SIZE

(xac) SKIP1

(xac) SKIP2

(xac) SKIP3

SORT

(xac) SORT-CORE-SIZE

(xac) SORT-FILE-SIZE

(xa) SORT-MERGE

(spn) SORT-MESSAGE

(xac) SORT-MODE-SIZE

(xac) SORT-OPTION

(xac) SORT-RETURN

SOURCE

(xac) SOURCE-COMPUTER

SPACE

SPACES

SPECIAL-NAMES

STANDARD

(xa) START

STATUS

STOP

(spn) STRING

(spn) SUB-QUEUE-1

(spn) SUB-QUEUE-2

(spn) SUB-QUEUE-3

SUBTRACT

SUM

(ca) SUPERVISOR

(xa) SUPPRESS

(ca) SUSPEND

(spn) SYMBOLIC

SYNC

(spn) SYNCHRONIZED

(spn) SYSIN

(sp) SYSIPT

(sp) SYSST

(spn) SYSOUT

(sp) SYSPCH

(sp) SYSPUNCH

(sp) S01

(sp) S02

(sp) S03

(sp) S04

(sp) S05

(spn) TABLE

TALLY

TALLYING

TAPE

(ca) TERMINAL

(ca) TERMINATE

(spn) TEXT

(xac) THAN

(ca) THROUGH

THRU

(spn) TIME

(xac) TIME-OF-DAY

TIMES

TO

(ca) TOP

(spn) TOTALED

(spn) TOTALING

(xac) TRACE

(spn) TRACK

(xac) TRACK-AREA

(spn) TRACK-LIMIT

(xac) TRACKS

(xa) TRAILING

(xac) TRANSFORM

TYPE

(ca) UNEQUAL

(spn) UNIT

(spn) UNSTRING

UNTIL

UP

(ca) UPON

(ca) UPPER-BOUND

(ca) UPPER-BOUNDS

(sp) UPSI-0

(sp) UPSI-1

(sp) UPSI-2

(sp) UPSI-3

(sp) UPSI-4

(sp) UPSI-5

(sp) UPSI-6

(sp) UPSI-7

USAGE

USE

USING

VALUE

VALUES

VARYING

(ca) WHEN

(xac) WHEN-COMPILED

WITH

WORDS

WORKING-STORAGE

WRITE

(xac) WRITE-ONLY

(xac) WRITE-VERIFY

ZERO

ZEROS

ZEROS

TRIM HERE

TRIM HERE

FOLD

FOLD

TRIM HERE

TRIM HERE

(ca) DEBUG-SUB-1 INITIATE
 (ca) DEBUG-SUB-2 INPUT
 (ca) DEBUG-SUB-3 INPUT-OUTPUT
 (ca) DEBUGGING (xac) INSERT
 DECIMAL-POINT (ca) INSPECT
 DECLARATIVES INSTALLATION
 (xa) DELETE INTO
 (spn) DELIMITED INVALID
 (spn) DELIMITER IS
 (spn) DEPENDENT
 (spn) DEPTH JUST
 (spn) DESCENDING JUSTIFIED
 (spn) DESTINATION
 ~ DETAIL
 (ca) DISABLE KEY
 (spn) DISP (spn) LABEL
 (xac) DISPLAY LABEL-RETURN
 (ca) DISPLAY-ST LAST
 (ca) DISPLAY-n LEADING
 DIVIDE (spn) LEAVE
 DIVISION (spn) LEFT
 DOWN (spn) LENGTH
 (ca) DUPLICATES (ca) LESS
 (xa) DYNAMIC (ca) LIBRARY
 (spn) EGI LIMITS
 (xac) EJECT (ca) LINAGE
 (ca) ELSE (ca) LINAGE-COUNTER
 (spn) EMH LINE
 (ca) ENABLE LINE-COUNTER
 END LINES
 (xa) END-OF-PAGE (xa) LINKAGE
 ENDING LOW-VALUE
 ENTER LOW-VALUES
 (xac) ENTRY
 ENVIRONMENT (xac) MASTER-INDEX
 (xa) EOP MEMORY
 (ca) EQUAL EQUALS
 (ca) ERROR (spn) MESSAGE
 (spn) ESI (ca) MODE
 EVERY (xac) MODULES
 EXAMINE (ca) MORE-LABELS
 (ca) EXCEEDS MOVE
 (xa) EXCEPTION MULTIPLY
 (xac) EXHIBIT MULTIPLY
 (xa) EXIT (xac) NAMED
 (xac) EXTEND NEGATIVE
 (xac) EXTENDED-SEARCH NEXT
 NO
 (xac) NOMINAL
 FILE NOT
 FILE-CONTROL NOTE
 FILE-LIMIT (xac) NSTD-REELS
 FILE-LIMITS NUMBER
 FILLER (ca) NUMERIC
 FINAL NUMERIC-EDITED
 FIRST
 FOOTING
 FOR (ca) OBJECT-COMPUTER
 FROM OBJECT-PROGRAM
 OF
 OCCURS
 OFF
 OMITTED
 ON
 (xac) GOBACK OPEN
 GREATER OPTIONAL
 GROUP (ca) OR
 (xac) ORGANIZATION
 HEADING OTHERWISE
 HIGH-VALUE OUTPUT
 HIGH-VALUES (spn) OVERFLOW
 HOLD
 I-O
 I-O-CONTROL (ca) PAGE
 ID (xac) PAGE-COUNTER
 IDENTIFICATION PASSWORD
 IF PERFORM
 IN PF
 INDEX PH
 (ca) INDEX-n PICTURE
 INDEXED (spn) PLUS
 INDICATE (spn) POINTER
 (ca) INITIAL (xac) POSITION
 (ca) INITIALIZE (xac) POSITIONING
 POSITIVE

PROCEDURE DIVISION — BASIC FORMATS

(PROCEDURE DIVISION: PROCEDURE DIVISION ASSIGNING NUMBER 1 TO IDENTIFIER 2)

ACCEPT Statement

ACCEPT identifier [FROM {CONSOLE | mnemonic-name}]

ADD Statement

FORMAT 1

ADD {identifier-1} [literal-1] {identifier-2} [literal-2] ... TO identifier-n [ROUNDED]

[identifier-n [ROUNDED]] ... [ON SIZE ERROR imperative-statement]

FORMAT 2

ADD {identifier-1} {identifier-2} [literal-1] {literal-2} [literal-3] ...

GIVING identifier-n [ROUNDED] [ON SIZE ERROR imperative-statement]

FORMAT 3

ADD {CORRESPONDING} {CORR} identifier-1 TO identifier-2 [ROUNDED]

[ON SIZE ERROR imperative-statement]

ALTER Statement

ALTER procedure-name-1 TO {PROCEED TO} procedure-name-2 [procedure-name-3 TO {PROCEED TO} procedure-name-J] ...

CLOSE Statement

CLOSE file-name-1 [REEL UNIT] [WITH {NO REWIND} | LOCK]

[file-name-2 [REEL UNIT] [WITH {NO REWIND} | LOCK]] ...

COMPUTE Statement

COMPUTE identifier-1 [ROUNDED] = {arithmetic-expression} [identifier-2] [literal-1]

[ON SIZE ERROR imperative-statement]

DECLARATIVE Section

PROCEDURE DIVISION.

DECLARATIVES.

{section-name SECTION. USE sentence.

{paragraph-name. {sentence} ...} ...

END DECLARATIVES.

DISPLAY Statement

DISPLAY {identifier-1} [literal-1] {identifier-2} [literal-2] ... [UPON {CONSOLE | mnemonic-name}]

TRIM HERE

FOLD

FOLD
TRIM HERE

DIVIDE Statement
 FORMAT 1
 DIVIDE {identifier-1} INTO identifier-2 [ROUNDED]
 {literal-1} {BY} {literal-2} {GIVING: identifier-3 [ROUNDED]}
 [ON SIZE ERROR imperative-statement]

FORMAT 2
 DIVIDE {identifier-1} {INTO} {identifier-2} {GIVING: identifier-3 [ROUNDED]}
 {literal-1} {BY} {literal-2} {REMAINDER identifier-4} [ON SIZE ERROR imperative-statement]

ENTER Statement
 ENTER language-name [routine-name].

[REDACTED]

EXAMINE Statement
 FORMAT 1
 EXAMINE identifier TALLYING {UNTIL FIRST
 {ALL
 {LEADING} } literal-1
 [REPLACING BY literal-2]

FORMAT 2
 EXAMINE identifier REPLACING {ALL
 {LEADING
 {FIRST
 {UNTIL FIRST} } literal-1 BY literal-2

EXIT Statement
 paragraph-name. EXIT [REDACTED].

[REDACTED]

GO TO Statement
 FORMAT 1
 GO TO procedure-name-1

FORMAT 2
 GO TO procedure-name-1 [procedure-name-2] ... DEPENDING ON identifier

GO TO
 GO TO.

IF Statement
 IF condition {NEXT SENTENCE} {statement-1} {ELSE} {NEXT SENTENCE} {statement-2}

MOVE Statement
 FORMAT 1
 MOVE {identifier-1} TO identifier-2 [identifier-3] ...

FORMAT 2
 MOVE {CORRESPONDING} {CORR} identifier-1 TO identifier-2

IBM AMERICAN NATIONAL STANDARD
 COBOL RESERVED WORDS

No word in the following list should appear as a programmer defined name. The keys that appear before some of the words, and their meanings, are:

- (xa) before a word means that the word is an IBM extension to American National Standard COBOL.
- (xac) before a word means that the word is an IBM extension to both American National Standard COBOL and CODASYL COBOL.
- (ca) before a word means that the word is a CODASYL COBOL reserved word not incorporated in American National Standard COBOL or in IBM American National Standard COBOL.
- (sp) before a word means that the word is an IBM function-name established in support of the SPECIAL-NAMES function.
- (spn) before a word means that the word is used by an IBM American National Standard COBOL compiler, but not this compiler.
- (asn) before a word means that the word is defined by American National Standard COBOL, but is not used by this compiler.

	ACCEPT	(xa)	COMP-1
	ACCESS	(xa)	COMP-2
	ACTUAL	(xa)	COMP-3
	ADD	(xa)	COMP-4
(asn)	ADDRESS	(xa)	COMPUTATIONAL
	ADVANCING	(xa)	COMPUTATIONAL-1
	AFTER	(xa)	COMPUTATIONAL-2
	ALL	(xa)	COMPUTATIONAL-3
(ca)	ALPHABETIC	(xa)	COMPUTATIONAL-4
(ca)	ALPHANUMERIC		COMPUTE
	ALPHANUMERIC-EDITED		CONFIGURATION
	ALTER	(sp)	CONSOLE
	ALTERNATE		CONTAINS
(xa)	AND		CONTROL
	APPLY		CONTROLS
	ARE		COPY
	AREA	(xac)	CORE-INDEX
	AREAS		CORR
	ASCENDING		CORRESPONDING
	ASSIGN	(spn)	COUNT
	AT	(sp)	CSP
	AUTHOR		CURRENCY
(xac)	BASIS	(xac)	CURRENT-DATE
	BEFORE	(xac)	CYL-INDEX
	BEGINNING	(xac)	CYL-OVERFLOW
	BLANK	(sp)	C01
	BLOCK	(sp)	C02
(ca)	BOTTOM	(sp)	C03
	BY	(sp)	C04
		(sp)	C05
		(sp)	C06
		(sp)	C07
(xa)	CALL	(sp)	C08
(spn)	CANCEL	(sp)	C09
(spn)	CBL	(sp)	C10
(spn)	CD	(sp)	C11
	CF	(sp)	C12
	CH		DATA
(xac)	CHANGED		DATE
(xa)	CHARACTER	(spn)	DATE-COMPILED
	CHARACTERS		DATE-WRITTEN
(asn)	CLOCK-UNITS		DAY
	CLOSE	(spn)	DAY-OF-WEEK
(asn)	COBOL	(ca)	DE
	CODE		DEBUG
(xac)	COLUMN	(xac)	DEBUG-CONTENTS
	COM-REG	(ca)	DEBUG-ITEM
(spn)	COMMA	(ca)	DEBUG-LINE
	COMMUNICATION	(ca)	DEBUG-NAME
	COMP		

TRIM HERE

FOLD

TRIM HERE

FOLD
TRIM HERE

DEBUGGING LANGUAGE - BASIC FORMATS

Procedure Division Debugging Formats

EXHIBIT Statement

EXHIBIT { NAMED CHANGED NAMED } { identifier-1 } { nonnumeric-literal-1 } [{ identifier-2 } { nonnumeric-literal-2 }] ...

ON (Count-conditional) Statement

FORMAT 1

ON integer-1 [AND EVERY integer-2] [UNTIL integer-3]
{ imperative-statement ... } { ELSE } { statement ... }
{ NEXT SENTENCE } { OTHERWISE } { NEXT SENTENCE }

FORMAT 2 (Version 3)

ON { integer-1 } { identifier-1 } [AND EVERY { integer-2 } { identifier-2 }]
[UNTIL { integer-3 } { identifier-3 }] { NEXT SENTENCE }
{ ELSE } { statement ... }
{ OTHERWISE } { NEXT SENTENCE }

READY/RESET TRACE Statement

{ READY } TRACE
{ RESET }

Compile-Time Debugging Packet

DEBUG Card

DEBUG location

FORMAT CONTROL - BASIC FORMATS

EJECT Statement

1 Area B

EJECT

SKIP1, SKIP2, SKIP3 Statements

1 Area B

{ SKIP1 }
{ SKIP2 }
{ SKIP3 }

STERLING CURRENCY - BASIC FORMATS

Data Division Sterling Formats

Nontreport PICTURE Clause

{ PICTURE } IS 9[(n)] D [8] 8D { 6[6] } [[V] 9 [(n)]] [USAGE IS] DISPLAY-ST

Report PICTURE Clause

{ PICTURE } IS [pound-report-string] [pound-separator-string] delimiter
shilling-report-string [shilling-separator-string] delimiter
pence-report-string [pence-separator-string] [sign-string]
[USAGE IS] DISPLAY-ST

MULTIPLY Statement

FORMAT 1

MULTIPLY { identifier-1 } { literal-1 } BY identifier-2 [ROUNDED]
[ON SIZE ERROR imperative-statement]

FORMAT 2

MULTIPLY { identifier-1 } { literal-1 } BY { identifier-2 } { literal-2 } GIVING identifier-3
[ROUNDED] [ON SIZE ERROR imperative-statement]

NOTE Statement

NOTE character string

OPEN Statement

OPEN [INPUT { file-name } [REVERSED WITH NO REWIND]] ...]
[OUTPUT { file-name } [WITH NO REWIND]] ...]
[IO { file-name } ...]

PERFORM Statement

FORMAT 1

PERFORM procedure-name-1 [THRU procedure-name-2]

FORMAT 2

PERFORM procedure-name-1 [THRU procedure-name-2] { identifier-1 } { integer-1 } TIMES

FORMAT 3

PERFORM procedure-name-1 [THRU procedure-name-2] UNTIL condition-1

FORMAT 4

PERFORM procedure-name-1 [THRU procedure-name-2]

VARYING { index-name-1 } { identifier-1 } FROM { index-name-2 } { literal-2 } BY
{ literal-3 } { identifier-3 } UNTIL condition-1

[AFTER { index-name-4 } { identifier-4 } FROM { index-name-5 } { literal-5 } BY
{ literal-6 } { identifier-6 } UNTIL condition-2

[AFTER { index-name-7 } { identifier-7 } FROM { index-name-8 } { literal-8 } BY
{ literal-9 } { identifier-9 } UNTIL condition-3]

READ Statement

READ file-name RECORD [INTO identifier] { AT END } { INVALID KEY } imperative-statement

REWRITE Statement

SEEK Statement

SEEK file-name RECORD

FOLD
TRIM HERE

FOLD

TRIM HERE



STOP Statement

STOP {RUN}
{literal}

SUBTRACT Statement

FORMAT 1

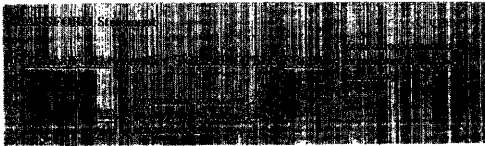
SUBTRACT {identifier-1} {literal-1} {identifier-2} {literal-2} ... FROM identifier-m [ROUNDED]
{identifier-n [ROUNDED]} ... [ON SIZE ERROR imperative-statement]

FORMAT 2

SUBTRACT {identifier-1} {literal-1} {identifier-2} {literal-2} ... FROM {identifier-m} {literal-m} GIVING
identifier-n [ROUNDED] [ON SIZE ERROR imperative-statement]

FORMAT 3

SUBTRACT {CORRESPONDING} {CORR} identifier-1 FROM identifier-2 [ROUNDED]
[ON SIZE ERROR imperative-statement]



USE Sentence

FORMAT 1

Option 1:

USE {BEFORE} STANDARD {BEGINNING} {REEL} {FILE} {UNIT} {AFTER}

LABEL PROCEDURE ON {file-name} ... {OUTPUT} {INPUT} {LO}

Option 2:

USE {BEFORE} STANDARD {ENDING} {REEL} {FILE} {UNIT} {AFTER}

LABEL PROCEDURE ON {file-name} ... {OUTPUT} {INPUT} {LO}

FORMAT 2

USE AFTER STANDARD ERROR PROCEDURE

ON {file-name-1} {INPUT} {OUTPUT} {LO}

NOTE: Format 3 of the USE Sentence is included in Formats for the REPORT WRITER feature.

SET Statement

FORMAT 1

SET {index-name-1} {index-name-2} ... TO {index-name-3} {identifier-3} {literal-1}

FORMAT 2

SET index-name-4 [index-name-5] ... {UP BY} {DOWN BY} {identifier-4} {literal-2}

SEGMENTATION — BASIC FORMATS

Environment Division Segmentation Formats

SEGMENT-LIMIT Clause

Object-Computer Paragraph

SEGMENT-LIMIT IS priority-number

Procedure Division Segmentation Formats

Priority Numbers

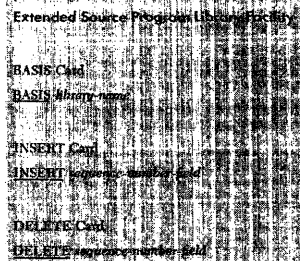
section-name SECTION [priority-number].

SOURCE PROGRAM LIBRARY FACILITY — BASIC FORMATS

COPY Statement

COPY library-name {ADDRESS}

{REPLACING} word-1 BY {word-2} {literal-1} {word-3} BY {word-4} {literal-2} {identifier-2}] ...]



TRIM HERE

TRIM HERE

FOLD

FOLD
TRIM HERE

Procedure Division Report Writer Formats

GENERATE Statement

GENERATE identifier

INITIATE Statement

INITIATE report-name-1 [report-name-2] ...

TERMINATE Statement

TERMINATE report-name-1 [report-name-2] ...

USE Sentence

USE BEFORE REPORTING identifier-1.

TABLE HANDLING - BASIC FORMATS

Data Division Table Handling Formats

OCCURS Clause

FORMAT 1

OCCURS integer-2 TIMES

[{ ASCENDING } KEY IS data-name-2 [data-name-3] ...]

[INDEXED BY index-name-1 [index-name-2] ...]

FORMAT 2

OCCURS integer-1 TO integer-2 TIMES [DEPENDING ON data-name-1]

[{ ASCENDING } KEY IS data-name-2 [data-name-3] ...]

[INDEXED BY index-name-1 [index-name-2] ...]

FORMAT 3

OCCURS integer-2 TIMES [DEPENDING ON data-name-1]

[{ ASCENDING } KEY IS data-name-2 [data-name-3] ...]

[INDEXED BY index-name-1 [index-name-2] ...]

USAGE Clause

[USAGE IS] INDEX

Procedure Division Table Handling Formats

SEARCH Statement

FORMAT 1

SEARCH identifier-1 [VARYING { identifier-1 }]

[AT END imperative-statement-1]

WHEN condition-1 { imperative-statement-2 }
NEXT SENTENCE

[WHEN condition-2 { imperative-statement-3 }
NEXT SENTENCE] ...]

FORMAT 2

SEARCH ALL identifier-1 [AT END imperative-statement-1]

WHEN condition-1 { imperative-statement-2 }
NEXT SENTENCE

WRITE Statement

FORMAT 1

WRITE record-name [FROM identifier-1] [{ BEFORE }
AFTER] ADVANCING

{ identifier-2 LINES }
integer LINES } [AT { END-OF-PAGE }
{ EOP } imperative-statement]

FORMAT 2

WRITE record-name [FROM identifier-1] AFTER POSITIONING { identifier-2 }
integer LINES

[AT { END-OF-PAGE }
{ EOP } imperative-statement]

FORMAT 3

WRITE record-name [FROM identifier-1] INVALID KEY imperative-statement

SORT - BASIC FORMATS

Environment Division Sort Formats

FILE-CONTROL PARAGRAPH - SELECT SENTENCE

SELECT Sentence (for GIVING option only)

SELECT file-name

ASSIGN TO [integer-1] system-name-1 [system-name-2] ...

OR system-name-3 [FOR MULTIPLE { REEL }
{ UNIT }]

[RESERVE { integer-2 }
{ NO }] ALTERNATE [AREA
AREAS]]

SELECT Sentence (for Sort Work Files)

SELECT sort-file-name

ASSIGN TO [integer] system-name-1 [system-name-2] ...

I-O CONTROL PARAGRAPH

RERUN Clause

RERUN ON system-name

SAME RECORD/SORT AREA Clause

SAME { RECORD }
{ SORT } AREA FOR file-name 1 [file-name-2] ...

Data Division Sort Formats

SORT-FILE DESCRIPTION

SD sort-file-name

RECORDING MODE IS mode

DATA { RECORD IS }
{ RECORDS ARE } data-name-1 [data-name-2] ...

RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS

LABEL { RECORD IS } { STANDARD }
{ RECORDS ARE } { OMITTED } ...

SORT-OPTION IS data-name-3. (DOS/VS COBOL only)

TRIM HERE

FOLD
TRIM HERE

FOLD

TRIM HERE

Procedure Division Sort Formats

RELEASE Statement

RELEASE sort-record-name [FROM identifier]

RETURN Statement

RETURN sort-file-name RECORD [INTO identifier]
AT END imperative-statement

SORT Statement

SORT file-name-1 ON {DESCENDING | ASCENDING} KEY {data-name-1} ...
[ON {DESCENDING | ASCENDING} KEY {data-name-2} ...] ...
{INPUT PROCEDURE IS section-name-1 [THRU section-name-2]}
{USING file-name-2}
{OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4]}
{GIVING file-name-3}

REPORT WRITER - BASIC FORMATS

Data Division - Report Writer Formats

NOTE: Formats which appear as Basic Formats within the general description of the Data Division are illustrated there.

FILE SECTION - REPORT Clause

{REPORT IS | REPORTS ARE} report-name-1 [report-name-2] ...

REPORT SECTION

REPORT SECTION:

RD report-name

WITH CODE mnemonic-name

{CONTROL IS | CONTROLS ARE} {FINAL identifier-1 [identifier-2] ... | FINAL identifier-1 [identifier-2] ...}

PAGE [LIMIT IS | LIMITS ARE] integer-1 {LINE | LINES}

[HEADING integer-2]

[FIRST DETAIL integer-3]

[LAST DETAIL integer-4]

[FOOTING integer-5]

REPORT GROUP DESCRIPTION ENTRY

FORMAT 1

01 [data-name-1]

LINE NUMBER IS {integer-1 PLUS integer-2 | NEXT PAGE}

NEXT GROUP IS {integer-1 PLUS integer-2 | NEXT PAGE}

TYPE IS {REPORT HEADING | RH | PAGE HEADING | PH | CONTROL HEADING | CH | DETAIL | DE | CONTROL FOOTING | CF | PAGE FOOTING | PF | REPORT FOOTING | RF} {identifier-n | FINAL}

USAGE Clause.

FORMAT 2

nn [data-name-1]

LINE Clause - See Format 1

USAGE Clause.

FORMAT 3

nn [data-name-1]

BLANK WHEN ZERO Clause

COLUMN NUMBER IS integer-1

GROUP INDICATE

JUSTIFIED Clause

LINE Clause - See Format 1

PICTURE Clause

RESET ON {identifier-1 | FINAL}

SOURCE IS {identifier-2}

SUM {literal-1} {literal-2} {literal-3} {literal-4} ... [UPON data-name]

VALUE IS literal-1

USAGE Clause.

FORMAT 4

01 data-name-1

BLANK WHEN ZERO Clause

COLUMN Clause - See Format 2

GROUP Clause - See Format 2

JUSTIFIED Clause

LINE Clause - See Format 1

NEXT GROUP Clause - See Format 1

PICTURE Clause

RESET Clause - See Format 2

{SOURCE Clause | SUM Clause | VALUE Clause} See Format 2

TYPE Clause - See Format 1

USAGE Clause.

TRIM HERE

TRIM HERE

FOLD

APPENDIX D: SUMMARY OF FILE-PROCESSING TECHNIQUES AND APPLICABLE STATEMENTS AND CLAUSES

This appendix summarizes the statements and clauses that may be specified for each file-processing technique. In addition, each file-name must be specified in a SELECT clause in the Environment Division and must be defined by an FD entry in the File Section of the Data Division.

STANDARD SEQUENTIAL FILES – Required and Optional Entries

Device Type	Required Entries					Optional Entries					
	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	RESERVE	ACCESS	Other ENVIRONMENT DIVISION Clauses	BLOCK CONTAINS ³	USE	
Reader	SYSnnn-UR-xxxx-S [-name]	OMITTED	INPUT	[LOCK]	READ [INTO] AT END	{integer} NO	SEQUENTIAL	SAME [RECORD] AREA RERUN	-	ERROR ⁴	
Punch	SYSnnn-UR-xxxx-S [-name]	OMITTED	OUTPUT	[LOCK]	WRITE ¹ [FROM] {BEFORE} ADVANCING {AFTER}	{integer} NO	SEQUENTIAL	SAME [RECORD] AREA RERUN	-	ERROR ⁴	
Printer	SYSnnn-UR-xxxx-S [-name]	OMITTED	OUTPUT	[LOCK]	WRITE ¹ [FROM] {BEFORE} ADVANCING {AFTER}	{integer} NO	SEQUENTIAL	SAME [RECORD] AREA RERUN	-	ERROR ⁴ REPORTING	
Tape	SYSnnn-UT-xxxx-S [-name]	{STANDARD OMITTED data-name }	INPUT	[REVERSED] [NO REWIND]	[REEL] [LOCK NO REWIND]	{integer} NO	SEQUENTIAL	SAME [RECORD] AREA RERUN MULTIPLE FILE TAPE	[n TO] m	LABEL ERROR	
			OUTPUT [NO REWIND]	[REEL] [LOCK NO REWIND]	WRITE ¹ [FROM] {BEFORE} ADVANCING {AFTER}					LABEL ERROR REPORTING	
Mass Storage	SYSnnn- $\begin{cases} UT \\ DA \end{cases}$ -xxxx-S [-name]	{STANDARD data-name }	INPUT		[UNIT] [LOCK]	{integer} NO	SEQUENTIAL	SAME [RECORD] AREA RERUN	[n TO] m	AFTER LABEL ERROR	
			OUTPUT		[UNIT] [LOCK]					WRITE ¹ [FROM] {BEFORE} ADVANCING {AFTER}	AFTER LABEL ERROR REPORTING
			I-O		[UNIT] [LOCK]					WRITE ¹ [FROM] INVALID KEY READ [INTO] AT END WRITE ² [FROM] INVALID KEY	AFTER LABEL ERROR

¹Create

²Update

³Not for U mode

⁴STXIT Option must be specified on the CBL control card

DIRECT FILES (mass storage devices only) – Required and Optional Entries

Required Entries							Optional Entries			
ACCESS	KEY	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	APPLY	Other ENVIRONMENT DIVISION Clauses	RECORDING MODE	USE
[SEQUENTIAL]	[ACTUAL]	SYSnnn-DA-xxxx- $\begin{Bmatrix} A \\ D \end{Bmatrix}$ [-name]	{STANDARD} {data-name }	INPUT	[UNIT] [LOCK]	READ [INTO] AT END		SAME [RECORD] AREA RERUN	$\begin{Bmatrix} F \\ U \\ S \end{Bmatrix}$	AFTER LABEL ERROR
RANDOM	ACTUAL	SYSnnn-DA-xxxx- $\begin{Bmatrix} A \\ D \end{Bmatrix}$ [-name]	{STANDARD} {data-name }	INPUT	[LOCK]	SEEK READ [INTO] INVALID KEY	EXTENDED SEARCH	SAME [RECORD] AREA RERUN	$\begin{Bmatrix} F \\ U \\ S \end{Bmatrix}$	AFTER LABEL ERROR
				OUTPUT	[LOCK]	SEEK WRITE ¹ [FROM] INVALID KEY	WRITE-VERIFY			
				I-O	[LOCK]	SEEK READ [INTO] INVALID KEY WRITE ² [FROM] INVALID KEY	EXTENDED SEARCH WRITE-VERIFY			
RANDOM	ACTUAL	SYSnnn-DA-xxxx- $\begin{Bmatrix} U \\ W \end{Bmatrix}$ [-name]	{STANDARD} {data-name }	INPUT	[LOCK]	SEEK READ [INTO] INVALID KEY	EXTENDED SEARCH	SAME [RECORD] AREA RERUN	$\begin{Bmatrix} F \\ U \\ S \end{Bmatrix}$	AFTER LABEL ERROR
				OUTPUT	[LOCK]	SEEK WRITE ² [FROM] INVALID KEY	WRITE-VERIFY			
RANDOM	ACTUAL	SYSnnn-DA-xxxx- $\begin{Bmatrix} U \\ W \end{Bmatrix}$ [-name]	{STANDARD} {data-name }	I-O	[LOCK]	SEEK READ [INTO] INVALID KEY	EXTENDED SEARCH	SAME [RECORD] AREA RERUN	$\begin{Bmatrix} F \\ U \\ S \end{Bmatrix}$	AFTER LABEL ERROR
						WRITE ³ [FROM] INVALID KEY	WRITE-VERIFY			
						REWRITE ⁴ [FROM] INVALID KEY	EXTENDED SEARCH WRITE-VERIFY			
				¹ Create	² Update and add	³ Add	⁴ Update			

PROCESSED FILES (magnetic storage devices only)		Required and Optional Entries												
		Required Entries					Optional Entries							
ACCESS	System name	KEY	LABEL RECORDS	OPEN	CLOSE	Access Verbs	APPLY	RESERVE	Other ENVIRONMENT DIVISION Clauses	RECORDING MODE	BLOCK CONTAINS	USE		
SEQUENTIAL	SYSname-DA-xxxx-1 [-name]	RECORD	STANDARD	INPUT	[LOCK]	READ [INTO]	[MASTER-INDEX] [CYL-INDEX]	NO	SAME [RECORD] AREA RERUN	F	m RECORDS	ERROR		
		RECORD NOMINAL				START INVALID KEY								
		RECORD				OUTPUT [LOCK]							WRITE [FROM] INVALID KEY	CYL-OVERFLOW [MASTER-INDEX] [CYL-INDEX]
		RECORD				IO [LOCK]							READ [INTO] AT END REWRITE ² [FROM] INVALID KEY	WRITE-VERIFY
		RECORD NOMINAL				START INVALID KEY								
RANDOM	SYSname-DA-xxxx-1 [-name]	RECORD	STANDARD	INPUT	[LOCK]	READ [INTO] INVALID KEY	CORE-INDEX [MASTER-INDEX] [CYL-INDEX]	NO	SAME [RECORD] AREA RERUN	F	m RECCRDS	ERROR		
		RECORD NOMINAL				IO [LOCK]							READ [INTO] INVALID KEY WRITE ³ [FROM] INVALID KEY REWRITE ² [FROM] INVALID KEY	CORE-INDEX CYL-OVERFLOW [MASTER-INDEX] [CYL-INDEX] WRITE-VERIFY

APPENDIX E: ASCII CONSIDERATIONS

This compiler supports the American National Standard Code for Information Interchange (ASCII). Thus the programmer can create and process tape files recorded in accordance with the following standards:

- ASCII Standard Code X3.4-1967
- American National Standard X3.27-1969, Magnetic Tape Labels for Information Interchange
- American National Standard 9-track, 800 bpi, NRZI Magnetic Tape Standard X3.22-1967

ASCII encoded tape files, when read into the system, are automatically translated in the buffers into EBCDIC. Internal manipulation of data is performed exactly as if they were EBCDIC encoded files. For an output file, the system translates the EBCDIC characters into ASCII in the buffers before writing the file out on tape. Therefore there are special considerations concerning ASCII encoded files when they are processed in COBOL. The following paragraphs discuss these considerations.

I -- ENVIRONMENT DIVISION

Environment Division clauses affected by the specification of ASCII files are the ASSIGN clause and the RERUN clause.

ASSIGN Clause

When ASCII files are to be processed, the system-name in the ASSIGN clause has the following format:

SYSnnn-UT-device-C[-offset][-name]

nnn is a three-digit number between 000 and 221. This number represents the symbolic unit to which the file is assigned.

UT for utility must be specified in the class field

device must specify a magnetic tape device (2400).

C in the organization field specifies that an ASCII encoded sequential file is to be processed, or that an ASCII collated sort is to be performed.

offset may be specified only for an ASCII file, and then only if a block prefix of length 01 through 99 exists. It is a 2-digit field, and may be specified as follows:

01 through 99	for an input file
04	for an output file (D-mode records only)

name is a one- to seven-character field specifying the external-name by which the file is known to the system. If specified, it is the name that appears in the file-name field of the VOL, DLBL, or TLBL job control statement. If this field is not specified, the symbolic unit (SYSnnn) is used as the external-name. This field must be specified if more than one file is assigned to the same symbolic unit.

RERUN Clause

The system-name in a RERUN clause must not specify an ASCII encoded file.

ASCII encoded files containing checkpoint records cannot be processed.

II -- DATA DIVISION

In the Data Division there are special considerations for ASCII files, both in the File Section and in Data Description Entries.

FILE SECTION

In the File Section the BLOCK CONTAINS clause, the LABEL RECORDS clause and the RECORDING MODE clause are affected. There are also special considerations regarding the compiler default options for recording mode.

BLOCK CONTAINS Clause

For an ASCII file that contains a buffer offset field, the following considerations apply:

- If the BLOCK CONTAINS clause with the RECORDS option is specified, or if the BLOCK CONTAINS clause is omitted, the compiler compensates for the buffer offset field.
- If the BLOCK CONTAINS clause with the CHARACTERS option is specified, the programmer must include the buffer offset as part of the physical record.

LABEL RECORDS Clause

All three options of the clause (OMITTED/STANDARD/data-name) are allowed. However, if the programmer specifies the data-name option, he must make sure that data-name refers only to user standard labels. Nonstandard labels are not allowed for ASCII files.

RECORDING MODE Clause

For ASCII files, mode may be specified as F, U, or V. S mode may not be specified.

Compiler Calculation of Recording Mode

When the RECORDING MODE clause is not used to specify the mode of the records in an ASCII file, the COBOL compiler determines the mode by scanning each record description entry. The default option may be:

- F if all the records are defined as being the same size.
- D if the records are defined as variable in size, or if the RECORD CONTAINS clause specifies variable size records. Internally D mode is the equivalent of V mode for EBCDIC encoded files.

DATA DESCRIPTION ENTRIES

For ASCII files the Data Description Entries affected are the PICTURE clause, the SIGN clause, and the USAGE clause.

PICTURE Clause

For ASCII files all five categories of data are valid.

If a data item is numeric, however, and the item is signed, then the SIGN clause with the SEPARATE CHARACTER option must also be specified.

SIGN Clause

If a data item in an ASCII file is numeric and has a sign, then the SIGN clause with the SEPARATE CHARACTER option must be specified.

USAGE Clause

For data items in ASCII files, only the DISPLAY option of the USAGE clause is valid.

III.-- PROCEDURE DIVISION

For ASCII files, there are special considerations in regard to Label Declaratives and relation conditions.

LABEL PROCEDURE Declarative

Since the user may not specify nonstandard labels for an ASCII encoded file, the BEFORE option of the LABEL PROCEDURE Declarative is not allowed.

Relation Conditions

If the ASCII character strings to be compared contain mixed alphabetic/numeric characters and/or special characters, then the TRANSFORM verb can be used before the comparison is made to ensure a valid comparison.

The following example illustrates a method of making the comparison (Figure 54 shows the necessary COBOL statements).

Suppose that the COBOL programmer specifies that one alphanumeric data item (ASCII-1) from ASCII-FILE is to be compared with another such data item (ASCII-2), and that the results of the comparison determine the path of program execution. Each data item may contain any valid COBOL character.

When ASCII-RECORD is read into the buffer, the system changes each ASCII character into its EBCDIC equivalent. Therefore, before a valid ASCII comparison can be made, the relative position of each character in the ASCII collating sequence must be reestablished.

In the Working-Storage Section, the VALUE of IDENT-EBCDIC is the ascending EBCDIC collating sequence (as shown in Figure 55); similarly, the VALUE of IDENT-ASCII is the ascending ASCII collating sequence. The contents of ASCII-1 are moved to DN-1, and the contents of ASCII-2 are moved to DN-2, and DN-1 and DN-2 are then used in the two TRANSFORM statements. (This avoids the necessity of a second pair of TRANSFORM statements to restore the original contents of ASCII-1 and ASCII-2.)

When the two TRANSFORM statements are executed, each EBCDIC character is exchanged for another EBCDIC character that occupies the original ASCII character's position in the ASCII collating sequence. Thus, when the comparison is made, it is valid for the ASCII collating sequence.

(Note that if ASCII-1 and ASCII-2 are restricted to mixed alphabetic and numeric characters, then the VALUE clauses in IDENT-EBCDIC and IDENT-ASCII need only contain alphabetic and numeric characters from the collating sequences. Note too that in the VALUE clause when quotation marks (") are used as delimiters, then the quotation mark itself cannot be one of the literals contained within the delimiter; similarly, if the apostrophe (') is used as the delimiter, then the apostrophe cannot be contained within the delimiter.)


```

DATA DIVISION.
FILE SECTION.
FD  ASCII-FILE
01  ASCII-RECORD
    05  ASCII-1
    05  ASCII-2
.
.
.
WORKING-STORAGE SECTION.
77  IDENT-ASCII  PICTURE X(50) VALUE
    " $(O)*+ , - / 0123456789 <=> ABCDEFGHIJKLMNOPQRSTUVWXYZ
77  IDENT-EBCDIC PICTURE X(50) VALUE
    " .<(+ $*) ; - / , > ' = ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
77  DN-1 .....
77  DN-2 .....
.
.
.
PROCEDURE DIVISION.
.
.
.
TEST-ASCII.
    MOVE ASCII-1 TO DN-1.
    MOVE ASCII-2 TO DN-2.
    TRANSFORM DN-1 FROM IDENT-EBCDIC TO IDENT-ASCII.
    TRANSFORM DN-2 FROM IDENT-EBCDIC TO IDENT-ASCII.
    IF DN-1 IS NOT LESS THAN DN-2
        PERFORM PROCESS-1
    ELSE PERFORM PROCESS-2.
.
.
.
PROCESS-1.
.
.
.
PROCESS-2.
.
.
.

```

Figure 54. Using the TRANSFORM Statement with ASCII Comparisons

<u>EBCDIC Collating Sequence</u>	<u>ASCII Collating Sequence</u>
1. (space)	1. (space)
2. . (period, decimal point)	2. " (quotation mark)
3. < (less than)	3. \$ (currency symbol)
4. ((left parenthesis)	4. ' (apostrophe, single quotation mark)
5. + (plus symbol)	5. ((left parenthesis)
6. \$ (currency symbol)	6.) (right parenthesis)
7. * (asterisk)	7. * (asterisk)
8.) (right parenthesis)	8. + (plus symbol)
9. ; (semicolon)	9. , (comma)
10. - (hyphen, minus symbol)	10. - (hyphen, minus symbol)
11. / (stroke, virgule, slash)	11. . (period, decimal point)
12. , (comma)	12. / (stroke, virgule, slash)
13. > (greater than)	
14. ' (apostrophe, single quotation mark)	13-22. 0 through 9
15. = (equal sign)	
16. " (quotation mark)	23. ; (semicolon)
	24. < (less than)
17-42. A through Z	25. = (equal sign)
	26. > (greater than)
43-52. 0 through 9	27-52. A through Z

Figure 55. EBCDIC and ASCII Collating Sequences for COBOL Characters -- in Ascending Order

IV -- SORT FEATURE

For ASCII-collated sorts, there are special considerations in the Environment Division and in the Data Division.

ENVIRONMENT DIVISION

For ASCII-collated sorts, there are special considerations for the ASSIGN clause ~~and for the DDNAME clause.~~

ASSIGN Clause

The ASSIGN clause for an ASCII collated sort has the same format as for an EBCDIC collated sort. However, the system-name must be in the following format:

SYSnnn-class-device-C-name

The following considerations apply:

SYSnnn must specify the fixed sort work units assigned to the sort-file. The first work unit for every sort-file in the program must be assigned to SYS001, the second to SYS002, etc.

class may be specified as UT or DA.

device may specify a utility or mass storage device.

C in the organization field specifies an ASCII collated sort.

name specifies the external-name by which the sort-file is known to the system. If the file has standard labels, the name field must be specified as SORTWK1 for SYS001, SORTWK2 for SYS002, etc.

Note: For an ASCII-collated sort, the buffer offset field is not permitted.

RERUN clause
Checkpoint records for ASCII-collated sorts can be taken. However, the system-name specified in the RERUN clause must not specify an ASCII-encoded file.

DATA DIVISION

For ASCII-collated sorts, there are special considerations for the SIGN clause and for the USAGE clause.

SIGN Clause
If an S is specified in the PICTURE of a numeric item to be used as a sort key in an ASCII-collated sort, the SEPARATE option of the SIGN clause must be specified.

USAGE Clause

If an ASCII-collated sort is requested, the sort keys must be DISPLAY items, explicitly or implicitly.

APPENDIX F: SYMBOLIC DEBUGGING FEATURE

A programmer using IBM Full American National Standard COBOL, Version 3, under the Disk Operating System, has several methods available to him for testing and debugging his programs. Use of the symbolic debugging feature is the easiest and most efficient method for testing and debugging and is described in detail in this appendix.

The symbolic debug option produces a symbolic formatted dump of the object program's data area when the program abnormally terminates. It also enables the programmer to request dynamic dumps of specific data-names at strategic points during program execution. If two or more COBOL programs are link edited together and one of them terminates abnormally, the program causing termination and any callers compiled with the symbolic debug option, up to and including the main program, will be given a formatted dump.

The abnormal termination dump consists of the following parts:

1. Abnormal termination message, including the number of the statement and of the verb being executed at the time of an abnormal termination.
2. Selected areas in the Task Global Table.
3. Formatted dump of the Data Division including:
 - (a) for an SD, the card number, the sort-file-name, the type, and the sort record.
 - (b) for an FD, the card number, the file-name, the type, SYSnnn, DTF status, the contents of the Pre-DTF and DTF in hexadecimal, and the fields of the record.
 - (c) for an RD, the card number, the report-name, the type, the report line, and the contents of PAGE-COUNTER and LINE-COUNTER if present.
 - (d) For an index-name, the name, the type, and the contents in decimal.

Note: For DTFDA when ACCESS IS RANDOM, the actual key is not provided in the Pre-DTF.

Operation of the symbolic debug option is dependent on object-time control cards placed in the input stream. These cards are discussed below.

Object-Time Control Cards

The operation of the symbolic debug option is determined by two types of control cards:

Program-control card -- required if abnormal termination and/or dynamic dumps are requested.

Line-control card -- required only if dynamic dumps are requested.

Program-Control Cards: A program-control card must be present at execution time for any program requesting symbolic debugging. A program-control card must contain the following information:

The 1-8 character program-name of the COBOL program compiled using symbolic debugging.

The logical unit and file-name assigned to the file produced at compile time on SYS005.

Additional optional parameters can also be specified:

An entry used to provide a trace of a program-name when several programs are link edited together. Each time the specified program is entered, its program name is displayed.

Two formats of the Data Division area in the abnormal termination dump are allowed:

1. Level-01 items are provided in hexadecimal. Items subordinate to level-01 items are printed in EBCDIC if possible. Level-77 items are printed both in hexadecimal and EBCDIC.
2. Level-77 items and items subordinate to level-01 items are provided in EBCDIC. If these items contain unprintable characters, hexadecimal notation is provided. This is the default option.

Line-Control Cards: A line-control card must contain the following information:

The card number associated with the point in the Procedure Division at which the dynamic dump is to be taken. The number specified is the compiler-generated card number.

Additional optional parameters can also be specified:

The position of the verb in the specified line number at which the dynamic dump is to be taken. When the verb position is not specified, the first verb in the line is assumed. Any verb position not exceeding 15 may be specified.

An equivalent to the COBOL statement "ON n AND EVERY m UNTIL k ...". This option limits the request dynamic dumps to specified times. For example "ON n" results in one dump, produced the *n*th time the line number is reached during execution. "ON n AND EVERY m" results in a dump the first time at the *n*th execution of the specified line number, and thereafter at every *m*th execution until end-of-job.

Two formats of the Data Division areas displayed in the dynamic dump are allowed:

1. Level-01 items are provided in hexadecimal. Items subordinate to level-01 items are provided in EBCDIC, if possible. Level-77 items are provided both in hexadecimal and EBCDIC.
2. Level-77 items subordinate to level-01 items are provided in EBCDIC. If these items contain unprintable characters, hexadecimal notation is provided. Note that if a group item is specified, neither the group nor the elementary items in the group are provided in hexadecimal. This is the default option.

Selected areas of the Data Division to be dumped. A single data-name or a range of consecutive data-names can be specified. (If the programmer wishes to see a subscripted item, he specifies the name of the item without the subscript; this results in a dump of every occurrence of the subscripted item.)

A dump of everything that would be dumped in the event of an abnormal termination can also be specified. This allows the programmer to receive a formatted dump at normal end-of-job. To do this, the programmer must specify the generated statement number of the STOP RUN, GOBACK, or EXIT PROGRAM statement.

Sample Program -- TESTRUN

Figure 57 is an illustration of a program that utilizes the symbolic debugging features. In the following description of the program and its output, letters identifying the text correspond to letters in the program listing.

- (A) Because the SYMDMP option is requested on the CBL card, the logical unit SYS005 must be assigned at compile time.
- (B) The CBL card specifications indicate that an alphabetically ordered cross-reference dictionary, a flow trace of 10 procedures, and the symbolic debug option are being requested.
- (C) An alphabetically ordered cross-reference dictionary of data-names and procedure-names is produced by the compiler as a result of the SXREF specification on the CBL card.
- (D) The file assigned at compile time to SYS005 to store SYMDMP information is assigned to SYS009 at execution time.
- (E) The SYMDMP control cards placed in the input stream at execution time are printed along with any diagnostics.
 - (1) The first card is the program-control card where:
 - (a) TESTRUN is the PROGRAM-ID.
 - (b) 9 is the logical unit to which the SYMDMP file is assigned.
 - (c) MT indicates that the SYMDMP file is on tape.
 - (d) (HEX) indicates the format of the abnormal termination dump.
 - (2) The second card is a line-control card which requests a (HEX) formatted dynamic dump of COUNT, NAME-FIELD, NO-OF-DEPENDENTS, and RECORD-NO prior to the first and every fourth execution of generated card number 71.
 - (3) The third card is also a line-control card which requests a (HEX) formatted dynamic dump of WORK-RECORD and B prior to the execution of generated card number 80.
- (F) The type code combinations used to identify data-names in abnormal termination and dynamic dumps are defined. Individual codes are illustrated in Figure 56.
- (G) The dynamic dumps requested by the first line-control card.

- Ⓜ The dynamic dumps requested by the second line-control card.
- Ⓜ Program interrupt information is provided by the system when a program terminates abnormally.
- Ⓜ The statement number information indicates the number of the verb and of the statement being executed at the time of the abnormal termination. The name of the program containing the statement is also provided.
- Ⓜ A flow trace of the last 10 procedures executed is provided because FLOW=10 was specified on the CBL card.
- Ⓜ Selected areas of the Task Global Table are provided as part of the abnormal termination dump.
- Ⓜ For each file-name, the generated card number, the file type, SYSnnn, the DTF status, and the fields of the Pre-DTF and DTF in hexadecimal are provided.
- Ⓜ The fields of records associated with each FD are provided in the format requested on the program-control card.
- Ⓜ The contents of the fields of the Working-Storage Section are provided in the format requested on the program-control card.
- Ⓜ The values associated with each of the possible subscripts are provided for data items described with an OCCURS clause.
- Ⓜ Asterisks appearing within the EBCDIC representation of the value of a given field indicate that the type and the actual content of the field conflict.

Note: When using the symbolic debugging option, level numbers appear "normalized" in the symbolic dump produced. For example, a group of data items described as:

```
01 RECORDA.
   05 FIELD-A.
     10 FIELD-A1 PIC X.
     10 FIELD-A2 PIC X.
```

will appear as follows in symbolic debugging output:

```
01 RECORDA...
02 FIELD-A...
03 FIELD-A1...
04 FIELD-A2...
```

Debugging TESTRUN

1. Referring to the statement number information J provided by the symbolic debug option, it is learned that theabend occurred during the execution of the first verb on card 80.
2. Generated card number 80 contains the statement COMPUTE B = B + 1.
3. Verifying the contents of B at the time of the abnormal termination R it can be seen that the usage of B (numeric packed) conflicts with the value contained in the data area reserved for B (numeric display).

4. The abnormal termination occurred while trying to perform an addition on a display item.

More complex errors may require the use of dynamic dumps to isolate the problem area. Line-control cards are included in TESTRUN merely to illustrate how they are used and the output they produce.

Code	Meaning
A	Alphabetic
B	Binary
D	Display
E	Edited
*	Subscripted Item
F	Floating Point
N	Numeric
P	Packed Decimal
S	Signed
OT	Overpunch Sign Trailing
OL	Overpunch Sign Leading
SL	Separate Sign Leading
ST	Separate Sign Trailing

Figure 56. Individual Type Codes Used in SYMDMP Output

```
// JOB DEBUGL
// OPTION NODECK, LINK, LIST, LISTX, SYM, ERRS
// ASSGN SYS005, X'183'
// EXEC FCOBOL
```

20.36.22



Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 1 of 11)

```

CBL SXREF, FLOW=10, SYMDMP, QUOTE, SEQ ← 8
00001 000010 IDENTIFICATION DIVISION.
00002 000020 PROGRAM-ID. TESTRUN.
00003 000030 AUTHOR. PROGRAMMER NAME.
00004 000040 INSTALLATION. NEW YORK DEVELOPMENT CENTER.
00005 000050 DATE-WRITTEN. APRIL 18, 1973.
00006 000060 DATE-COMPILED. 04/27/73.
00007 000070 REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008 000080 COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK
00009 000090 AS INPUT.
00010 000100
00011 000110 ENVIRONMENT DIVISION.
00012 000120 CONFIGURATION SECTION.
00013 000130 SOURCE-COMPUTER. IBM-360-H50.
00014 000140 OBJECT-COMPUTER. IBM-360-H50.
00015 000150 INPUT-OUTPUT SECTION.
00016 000160 FILE-CONTROL.
00017 000170 SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018 000180 SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019 000190
00020 000200 DATA DIVISION.
00021 000210 FILE SECTION.
00022 000220 FD FILE-1
00023 000230 LABEL RECORDS ARE OMITTED
00024 000240 BLOCK CONTAINS 5 RECORDS
00025 000250 RECORDING MODE IS F
00026 000255 RECORD CONTAINS 20 CHARACTERS
00027 000260 DATA RECORD IS RECORD-1.
00028 000270 01 RECORD-1.
00029 000290 05 FIELD-A PIC X(20).
00030 000290 FD FILE-2
00031 000300 LABEL RECORDS ARE OMITTED
00032 000310 BLOCK CONTAINS 5 RECORDS
00033 000320 RECORD CONTAINS 20 CHARACTERS
00034 000330 RECORDING MODE IS F
00035 000340 DATA RECORD IS RECORD-2.
00036 000350 01 RECORD-2.
00037 000370 05 FIELD-A PIC X(20).
00038 000370 WORKING-STORAGE SECTION.
00039 000380 01 FILLER.
00040 000400 02 KOUNT PIC S99 COMP SYNC.
00041 000410 02 ALPHABET PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00042 000420 02 ALPHA REDEFINES ALPHABET PIC X OCCURS 26 TIMES.
00043 000420 02 NUMBR PIC S99 COMP SYNC.
00044 000440 02 DEPENDENTS PIC X(26) VALUE "012340123401234012340".
00045 000450 02 DEPEND REDEFINES DEPENDENTS PIC X OCCURS 26 TIMES.
00046 000450 01 WORK-RECORD.
00047 000460 05 NAME-FIELD PIC X.
00048 000470 05 FILLER PIC X.
00049 000480 05 RECORD-NO PIC 9999.
00050 000490 05 FILLER PIC X VALUE IS SPACE.
00051 000510 05 LOCATION PIC AAA VALUE IS "NYC".
00052 000510 05 FILLER PIC X VALUE IS SPACE.
00053 000520 05 NO-OF-DEPENDENTS PIC XX.
00054 000540 05 FILLER PIC X(7) VALUE IS SPACES.
00055 000550 01 RECORDA.
00056 000560 02 A PICTURE S9(4) VALUE 1234.
00057 000570 02 B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00058 000550 PROCEDURE DIVISION.
00059 000590 BEGIN.
00060 000600 NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE
00061 000610 TO BE CREATED AND INITIALIZES THE COUNTERS.
00062 000620 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT, NUMBR.
00063 000630 NOTE THAT THE FOLLOWING CREATES INTERNALLY THE
00064 000640 RECORDS TO BE CONTAINED IN THE FILE, WRITES THEM
00065 000650 ON TAPE, AND DISPLAYS THEM ON THE CONSOLE.
00066 000660 STEP-2. ADD 1 TO KOUNT, NUMBR. MOVE ALPHA (KOUNT) TO
00067 000670 NAME-FIELD.
00068 000680 MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
00069 000660 MOVE NUMBR TO RECORD-NO.
00070 000700 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE.
00071 000710 WRITE RECORD-1 FROM WORK-RECORD.
00072 000720 STEP-4. PERFORM STEP-2 THRU STEP-3
00073 000730 UNTIL KOUNT IS EQUAL TO 26.
00074 000740 NOTE THAT THE FOLLOWING CLOSES THE OUTPUT FILE
00075 000750 AND REOPENS IT AS INPUT.
00076 000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00077 000770 NOTE THAT THE FOLLOWING READS BACK THE FILE
00078 000780 AND SINGLES OUT EMPLOYEES WITH NO DEPENDENTS.
00079 000790 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00080 000800 COMPUTE B = B + 1.
00081 000810 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00082 000820 NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD.
00083 000830 GO TO STEP-6.
00084 000780 STEP-8. CLOSE FILE-2.
00085 000790 STOP RUN.

```

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 2 of 11)

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=1-148	FD	FILE-1	DTF=01		DNM=1-148		DTFMT				F
DNM=1-179	01	RECORD-1	BL=1	000	DNM=1-179	DS 0CL20	GROUP				
DNM=1-200	02	FIELD-A	BL=1	000	DNM=1-200	DS 20C	DISP				
DNM=1-217	PD	FILE-2	DTF=02		DNM=1-217		DTFMT				F
DNM=1-248	01	RECORD-2	BL=2	000	DNM=1-248	DS 0CL20	GROUP				
DNM=1-269	02	FIELD-A	BL=2	000	DNM=1-269	DS 20C	DISP				
DNM=1-289	01	FILLER	BL=3	000	DNM=1-289	DS 0CL56	GROUP				
DNM=1-308	02	KOUNT	BL=3	000	DNM=1-308	DS 1H	COMP				
DNM=1-323	02	ALPHABET	BL=3	002	DNM=1-323	DS 26C	DISP				
DNM=1-341	02	ALPHA	BL=3	002	DNM=1-341	DS 1C	DISP			R	O
DNM=1-359	02	NUMBR	BL=3	01C	DNM=1-359	DS 1H	COMP				
DNM=1-374	02	DEPENDENTS	BL=3	01E	DNM=1-374	DS 26C	DISP				
DNM=1-394	02	DEPEND	BL=3	01E	DNM=1-394	DS 1C	DISP			R	O
DNM=1-410	01	WORK-RECORD	BL=3	038	DNM=1-410	DS 0CL20	GROUP				
DNM=1-434	02	NAME-FIELD	BL=3	038	DNM=1-434	DS 1C	DISP				
DNM=1-454	02	FILLER	BL=3	039	DNM=1-454	DS 1C	DISP				
DNM=1-473	02	RECORD-NO	BL=3	03A	DNM=1-473	DS 4C	DISP-NM				
DNM=1-492	02	FILLER	BL=3	03E	DNM=1-492	DS 1C	DISP				
DNM=2-000	02	LOCATION	BL=3	03F	DNM=2-000	DS 3C	DISP				
DNM=2-018	02	FILLER	BL=3	042	DNM=2-018	DS 1C	DISP				
DNM=2-037	02	NO-OF-DEPENDENTS	BL=3	043	DNM=2-037	DS 2C	DISP				
DNM=2-063	02	FILLER	BL=3	045	DNM=2-063	DS 7C	DISP				
DNM=2-082	01	RECORDA	BL=3	050	DNM=2-082	DS 0CL4	GROUP				
DNM=2-102	02	A	BL=3	050	DNM=2-102	DS 4C	DISP-NM				
DNM=2-113	02	B	BL=3	050	DNM=2-113	DS 4P	COMP-3			R	

MEMORY MAP

TGT	00400
SAVE AREA	00400
SWIICH	00448
TALLY	0044C
SORT SAVE	00450
ENTRY-SAVE	00454
SORT CORE SIZE	00458
NSTD-REELS	0045C
SORT RET	0045E
WORKING CELLS	00460
SORT FILE SIZE	00590
SORT MODE SIZE	00594
PGT-VN TBL	00598
TGT-VN TBL	0059C
SORTAB ADDRESS	005A0
LENGTH OF VN TBL	005A4
LNPTH OF SORTAB	005A6
PGM ID	005A8
A(INIT1)	005B0
UPSI SWITCHES	005B4
DEBUG TABLE PTR	005BC
CURRENT PRIORITY	005C0
TA LENGTH	005C1
PROCEDURE BLOCK1 PTR	005C4
UNUSED	005C8
OVERFLOW CELLS	005CC
BL CELLS	005CC
DIFADR CELLS	005D8
TEMP STORAGE	005E0
TEMP STORAGE-2	005E8
TEMP STORAGE-3	005E8
TEMP STORAGE-4	005E8
BLL CELLS	005E8
VLC CELLS	005EC
SBL CELLS	005EC
INDEX CELLS	005EC
SUBADR CELLS	005EC
ONCTL CELLS	005F4
PFMCTL CELLS	005F4
PFMSAV CELLS	005F4
VN CELLS	005F8
SAVE AREA =2	005FC
XSASW CELLS	005FC
XSA CELLS	005FC
PARAM CELLS	005FC
RPTSAV AREA	00600
CHECKPT CTR	00600
IOPTR CELLS	00600

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 3 of 11)

LITERAL POOL (HEX)

00668 (LIT+0) 00000001 001A1C5B 5BC2D6D7 C5D5405B 5BC2C3D3 D6E2C55B
 00680 (LIT+24) 5BC2C6C3 D4E4D300 C0000000

DISPLAY LITERALS (BCD)

0068C (LIT+36) 'WORK-RECORD'

PGT	00610
DEBUG LINKAGE AREA	00610
OVERFLOW CELLS	00618
VIRTUAL CELLS	00618
PROCEDURE NAME CELLS	0063C
GENERATED NAME CELLS	0064C
SUBDTF ADDRESS CELLS	00660
VNI CELLS	00660
LITERALS	00668
DISPLAY LITERALS	0068C
PROCEDURE BLOCK CELLS	00698

REGISTER ASSIGNMENT

REG 6 BL =3
 REG 7 BL =1
 REG 8 BL =2

WORKING-STORAGE STARTS AT LOCATION 00100 FOR A LENGTH OF 00058.

59	000698		START	EQU	*	
	000698	58 F0 C 018		L	15,018(0,12)	V(ILBDDBG4)
	00069C	05 EF		BALR	14,15	
	00069E	58 F0 C 01C		L	15,01C(0,12)	V(ILBDFLW1)
	0006A2	05 1F		BALR	1,15	
	0006A4	003B		DC	X'003B'	
62	0006A6	58 F0 C 018		L	15,018(0,12)	V(ILBDDBG4)
	0006AA	05 EF		BALR	14,15	
	0006AC	58 F0 C 01C		L	15,01C(0,12)	V(ILBDFLW1)
	0006B0	05 1F		BALR	1,15	
	0006B2	003E		DC	X'003E'	
62	0006B4	58 F0 C 018		L	15,018(0,12)	V(ILBDDBG4)
	0006B8	05 EF		BALR	14,15	
	0006BA	58 20 D 1CC		L	2,1CC(0,13)	BL=1
	0006BE	41 10 C 05F		LA	1,05F(0,12)	LIT+7
	0006C2	58 00 D 1D8		L	0,1D8(0,13)	DTF=1
	0006C6	18 40		LR	4,0	
	0006C8	07 00		BCR	0,0	
	0006CA	05 F0		BALR	15,0	
	0006CC	50 00 F 008		ST	0,008(0,15)	

STATISTICS	SOURCE RECORDS =	85	DATA ITEMS =	25	NO OF VERBS =	29
STATISTICS	PARTITION SIZE =	495560	LINE COUNT =	56	BUFFER SIZE =	256
OPTIONS IN EFFECT	PMAP RELOC ADR =	NONE	SPACING =	1	FLOW =	10
OPTIONS IN EFFECT	LISTX	QUOTE	SYM	NOCATALR	LIST	LINK
OPTIONS IN EFFECT	NOCLIST	FLAGW	ZWE	NOSUPMAP	NOXREF	ERRS
OPTIONS IN EFFECT	NOSTATE	TRUNC	SEQ	SYMDMP	NODECK	
					NGSTXII	LIB
					SXREF	NOOPT

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 4 of 11)


```

04/27/73  PHASE  XFR-AD  LOCORE  HICORE  DSK-AD  ESD TYPE  LABEL      LOADED  REL-FR
          PHASE*** 007000  007000  009BD3  20 03 1  CSECT     TESTRUN   007000  007000
          CSECT     IJFFBZZN  007AC8  007AC8
          * ENTRY   IJFFZZZN  007AC8
          * ENTRY   IJFFBZZZ  007AC8
          * ENTRY   IJFFZZZZ  007AC8
          CSECT     ILBDSAE0  009A98  009A98
          ENTRY     ILBDSAE1  009AAE
          CSECT     ILBDMNS0  009A90  009A90
          CSECT     ILBDDBG0  007F98  007F98
          ENTRY     ILBDDBG5  00847A
          ENTRY     ILBDDBG4  0084EC
          ENTRY     ILBDDBG7  008510
          ENTRY     ILBDDBG2  008262
          * ENTRY   ILBDDBG1  0080F4
          * ENTRY   ILBDDBG3  0084E2
          * ENTRY   ILBDDBG6  0084FC
          ENTRY     STXITPSW  0085A8
          * ENTRY   SORTEP    008748
          CSECT     ILBDFLW0  0095A0  0095A0
          ENTRY     ILBDFLW1  009660
          ENTRY     ILBDFLW2  00973C
          CSECT     ILBDIML0  009A38  009A38
          CSECT     ILBDADR0  007E38  007E38
          * ENTRY   ILBDADR1  007E44
          CSECT     ILBDDSP0  008DC8  008DC8
          * ENTRY   ILBDDSS0  008DC8
          * ENTRY   ILBDDSS1  009318
          * ENTRY   ILBDDSS2  0093B0
          * ENTRY   ILBDDSS3  009568
          CSECT     IJJCPDV1  0089F0  0089F0
          * ENTRY   IJJCPDV2  0089F0
          CSECT     IJJCPD1   008BD0  008BD0
          ENTRY     IJJCPD1N  008BD0
          * ENTRY   IJJCPD3   008BD0
          * UNREFERENCED SYMBOLS
          WXTRN     ILBDSTN0
          WXTRN     ILBDSRT0
          WXTRN     ILBDTEF3
          003 UNRESOLVED ADDRESS CONSTANTS
          // ASSGN SYS008,X'182'
          // ASSGN SYS009,X'183'
          // EXEC

```

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 6 of 11)

SYMDMP CONTROL CARDS

- ① TESTRUN,009,MT,(HEX)
- ② 71,ON 1,4,(HEX),KOUNT,NAME-FIELD,NO-OF-DEPENDENTS,RECORD-NO
- ③ 80,(HEX),WORK-RECORD,B

} ⑥

NO ERRORS FOUND IN CONTROL CARDS

⑦

TYPE CODES USED IN SYMDMP OUTPUT

CODE	MEANING
A	= ALPHABETIC
AN	= ALPHANUMERIC
ANE	= ALPHANUMERIC EDITED
D	= DISPLAY (STERLING NONREPORT)
DE	= DISPLAY EDITED (STERLING REPORT)
F	= FLOATING POINT (COMP-1/COMP-2)
FD	= FLOATING POINT DISPLAY (EXTERNAL FLOATING POINT)
NE	= NUMERIC BINARY UNSIGNED (COMP)
NB-S	= NUMERIC BINARY SIGNED
ND	= NUMERIC DISPLAY UNSIGNED (EXTERNAL DECIMAL)
ND-OL	= NUMERIC DISPLAY OVERPUNCH SIGN LEADING
ND-OT	= NUMERIC DISPLAY OVERPUNCH SIGN TRAILING
ND-SL	= NUMERIC DISPLAY SEPARATE SIGN LEADING
ND-ST	= NUMERIC DISPLAY SEPARATE SIGN TRAILING
NE	= NUMERIC EDITED
NP	= NUMERIC PACKED DECIMAL UNSIGNED (COMP-3)
NP-S	= NUMERIC PACKED DECIMAL SIGNED
*	= SUBSCRIPTED

⑧

TESTRUN LOC	AT CARD CARD	000071 LV NAME	TYPE	VALUE
007100	000040	02 KOUNT	NE-S (HEX)	+01 0001
007138	000047	02 NAME-FIELD	AN	A
007143	000053	02 NO-OF-DEPENDENTS	AN	0
00713A	000049	02 RECORD-NO	ND	0001
TESTRUN LOC	AT CARD CARD	000071 LV NAME	TYPE	VALUE
007100	000040	02 KOUNT	NE-S (HEX)	+05 0005
007138	000047	02 NAME-FIELD	AN	E
007143	000053	02 NO-OF-DEPENDENTS	AN	4
00713A	000049	02 RECORD-NO	ND	0005
TESTRUN LOC	AT CARD CARD	000071 LV NAME	TYPE	VALUE
007100	000040	02 KOUNT	NE-S (HEX)	+09 0009
007138	000047	02 NAME-FIELD	AN	I
007143	000053	02 NO-OF-DEPENDENTS	AN	3
00713A	000049	02 RECORD-NO	ND	00C9

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 7 of 11)

TESTRUN LOC	AT CARD CARD	000071 LV NAME	TYPE	VALUE
007100	000040	02 KOUNT	(HEX) NB-S	+13 000D
007138	000047	02 NAME-FIELD	AN	M
007143	000053	02 NO-OF-DEPENDENTS	AN	2
00713A	000049	02 RECORD-NO	ND	0013
TESTRUN LOC	AT CARD CARD	000071 LV NAME	TYPE	VALUE
007100	000040	02 KOUNT	(HEX) NB-S	+17 0011
007138	000047	02 NAME-FIELD	AN	Q
007143	000053	02 NO-OF-DEPENDENTS	AN	1
00713A	000049	02 RECORD-NO	ND	0017
TESTRUN LOC	AT CARD CARD	000071 LV NAME	TYPE	VALUE
007100	000040	02 KOUNT	(HEX) NB-S	+21 0015
007138	000047	02 NAME-FIELD	AN	U
007143	000053	02 NO-OF-DEPENDENTS	AN	0
00713A	000049	02 RECORD-NO	ND	0021
TESTRUN LOC	AT CARD CARD	000071 LV NAME	TYPE	VALUE
007100	000040	02 KOUNT	(HEX) NB-S	+25 0019
007138	000047	02 NAME-FIELD	AN	Y
007143	000053	02 NO-OF-DEPENDENTS	AN	4
00713A	000049	02 RECORD-NO	ND	0025
(H)				
TESTRUN LOC	AT CARD CARD	000080 LV NAME	TYPE	VALUE
007138	000046	01 WORK-RECORD	(HEX)	C1C1F0F0 F0F140D5 E8C340F0 40404040 40404040
007138	000047	02 NAME-FIELD	AN	A
007139	000048	02 FILLER	AN	A
00713A	000049	02 RECORD-NO	ND	0001
00713L	000050	02 FILLER	AN	
00713F	000051	02 LOCATION	A	NYC
007142	000052	02 FILLER	AN	
007143	000053	02 NO-OF-DEPENDENTS	AN	0
007145	000054	02 FILLER	AN	
007150	000057	02 B	(HEX) NE-S	*1*2*3* F1F2F3C4

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 8 of 11)

COBOL ABEND DIAGNOSTIC AIDS

I, INTERRUPT CODE 7 LAST PSW ADDR BEFORE ABEND D000790E ← (I)
 PROGRAM TESTRUN
 LAST CARD NUMBER/VERB NUMBER EXECUTED -- CARD NUMBER 000080/VERB NUMBER 01. (J)
 FLOW TRACE (K)
 TESTRUN 000066 000070 000066 000070 000066 000070 000066 000070 000076 000079

DATA DIVISION DUMP OF TESTRUN

(L)

TASK GLOBAL TABLE	LOC	VALUE
SAVE AREA	007400	00000000 01050108 00009998 800078EE 00007908 4000790A 000071E8 00007330
	007420	0000001A 000071E8 500079FA 00007100 00007274 00007330 000079C4 00007000
	007440	4000790E 00007610
SWITCH	007448	3C00004B
TALLY	00744C	00000000
SORT-SAVE	007450	00000000
ENTRY-SAVE	007454	00007698
SORT-CORE-SIZE	007458	00000000
NSTD-REELS	00745C	0000
SORT-RETURN	00745E	7E10
WORKING CELLS	007460	000071E8 000072C8 0000001A 000071E8 500079FA 00000000 FF000100 E9C1F0F0
	007480	F2F640D5 E8C340F0 40404040 40404040 00007F10 8000CEB0 0700746A 40000000
	0074A0	3100746C 40000005 080074A0 00000000 06007E10 60000101 920074C0 00000008
	0074C0	00000000 00000000 00000101 00000000 00000000 00000000 00000000 00000000
	0074E0	00008204 0000G103 00007548 00000000 0400ABE0 2008C4D2 E2D8F0F1 0G0077E2
	007500	0000001A 01000800 000077D0 00008DC8 00007137 000077E4 0000001A 0000001A
	007520	00007137 500079FA 00007100 00007260 00007330 000079C4 00007000 700077EA
	007540	00007610 8400CEB0 0700751A 40000006 3100751C 40000005 00027450 00007478
	007560	1D007E08 00000109 3100751C 40000005 08007568 00000000 1E007578 30000001
	007580	1C7ECBE3 8B030141 1E130B61 5C10C6E2
SORT-FILE-SIZE	007590	00000000
SORT-MODE-SIZE	007594	00000000
PGT-VN TBL	007598	C4E2D7F0
TGT-VN TBL	00759C	F3F090EC
SORTAB ADDR	0075A0	D10896FF
VN TBL LENGTH	0075A4	F7C9
SORTAB LENGTH	0075A6	D201
PROGRAM-ID	0075A8	TESTRUN
A(INIT1)	0075B0	00007000
UPSI-SWITCHES	0075B4	47F0F030 416D007C
TGT-DBS TABLE	0075BC	00000200
CURRENT PRIORITY	0075C0	00
TRANSIENT AREA LENGTH	0075C1	00D104
PROCEDURE-BLOCK	0075C4	91801000
UNUSED	0075C8	41110002
OVERFLOW CELLS	(NONE)	
BL CELLS	0075CC	00007274 00007330 00007100
DTFADR CELLS	0075D8	00007170 000071E8
TEMP STORAGE	0075E0	00000000 0000026C
BLL CELLS	0075E8	00000000
VLC CELLS	(NONE)	
SBL CELLS	(NONE)	
INDEX CELLS	(NONE)	
OTHER (SEE MEMORY MAP)	0075EC	0000711B 00007137 0000780C 0000780C 4780F1FA 0A000A40 000009AA 5F431B75
	00760C	06704450

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 9 of 11)

DATA DIVISION DUMP OF TESTRUN

LOC	CARD	LV NAME	TYPE	VALUE
	000017	FD FILE-1	(M)	STANDARD SEQUENTIAL, ASSIGNED TO SYS008, CLOSED
007158			PRE-DTF	01010014 00000000 00000000 00000000 0C0000C0 00000600
007170			DTFMT	00009200 0C000108 000071A8 000071B0 00007AC8 1160E2E8 E2F0FCF8 40400162
007190				00000000 00000000 00000000 86BCF018 41E0E001 58201044 01007260 20000064
0071B0				000072C8 000072C8 00000014 0000732B 00640063 00000000 00000000 C3009A98
0071D0				01010014 00000000 00000000 00000000 0C000000 00000000 00000200 0C000108
007274	000028	01 RECORD-1	(N)	
007274	000029	02 FIELD-A	(HEX) AN	D8C1F0F0 F1F740D5 E8C340F1 40404040 40404040 QA0017 NYC 1
	000018	FD FILE-2	(M)	STANDARD SEQUENTIAL, ASSIGNED TO SYS008, OPEN INPUT
0071D0			PRE-DTF	01010014 00000000 00000000 00000000 0C000000 00000000
0071E8			DTFMT	00008200 0C000108 00007220 00007228 00007AC8 11E8E2E8 E2F0F0F8 40400272
007208				10000000 24007902 00000001 86BCF018 41E0E001 58201044 02007398 00000064
007228				00007330 00007330 00000014 00007393 00640063 00000000 00009AAE 00009A98
007248				00000000 00000000 00000000 00000000 07000700 07000700 E9C1F0F0 F2F640D5
007330	000036	01 RECORD-2	(N)	
007330	000037	02 FIELD-A	(HEX) AN	C1C1F0F0 F0F140D5 E8C340F0 40404040 40404040 AA0001 NYC 0
	000039	01 FILLER	(P)	
007100			(HEX)	001AC1C2 C3C4C5C6 C7C8C9D1 D2D3D4D5 D6D7D8D9 E2E3E4E5
007118				E6E7E8E9 001AF0F1 F2F3F4F0 F1F2F3F4 F0F1F2F3 F4F0F1F2
007130				F3F4F0F1 F2F3F4F0
007100	000040	02 KOUNT	NE-S	+26
007102	000041	02 ALPHABET	AN	ABCDEFGHIJKLMNPOQRSTUVWXYZ
007102	000042	02 ALPHA	*AN	
		(Q) (SUB1)		
007102		1		A
007103		2		B
007104		3		C
007105		4		D
007106		5		E
007107		6		F
007108		7		G
007109		8		H
00710A		9		I
00710B		10		J
00710C		11		K
00710D		12		L
00710E		13		M
00710F		14		N
007110		15		O
007111		16		P
007112		17		Q

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 10 of 11)

DATA DIVISION DUMP OF TESTRUN

LOC	CARD	LV NAME	TYPE	VALUE
007113		18		R
007114		19		S-
007115		20		T
007116		21		U
007117		22		V
007118		23		W
007119		24		X
00711A		25		Y
00711B		26		Z
00711C	000043	02 NUMBR	NB-S	+26
00711E	000044	02 DEPENDENTS	AN	01234012340123401234012340
	000045	02 DEPEND	*AN	
		Ⓚ (SUB1)		
00711E		1		0
00711F		2		1
007120		3		2
007121		4		3
007122		5		4
007123		6		0
007124		7		1
007125		8		2
007126		9		3
007127		10		4
007128		11		0
007129		12		1
00712A		13		2
00712E		14		3
00712C		15		4
00712D		16		0
00712E		17		1
00712F		18		2
007130		19		3
007131		20		4
007132		21		0
007133		22		1
007134		23		2
007135		24		3
007136		25		4
007137		26		0
		Ⓟ		
007138	000046	01 WORK-RECORD	(HEX)	C1C1F0F0 F0F140D5 E8C340F0 40404040 40404040
007138	000047	02 NAME-FIELD	AN	A
007139	000048	02 FILLER	AN	A
00713A	000049	02 RECORD-NO	ND	0001
00713E	000050	02 FILLER	AN	
00713F	000051	02 LOCATION	A	NYC
007142	000052	02 FILLER	AN	
007143	000053	02 NO-OF-DEPENDENTS	AN	0
007145	000054	02 FILLER	AN	
		Ⓟ		
	000055	01 RECORDA	(HEX)	F1F2F3C4
007150	000056	02 A	ND-OT	+1234
007150	000057	02 B	Ⓡ → NP-S	*1*2*3*

END OF COBOL DIAGNOSTIC AIDS

Figure 57. Using the Symbolic Debugging Features to Debug the Program TESTRUN (Part 11 of 11)

APPENDIX G: COMBINED FUNCTION CARD PROCESSING

The IBM System/370 card punch devices offer more flexible processing capabilities than former card devices. When equipped with appropriate special features, these devices can be used separately as a card reader, as a card punch, or as a card printer. Any two or all three of these functions can be combined, so that those functions specified are all performed during one pass of a card through the device.

For any one data card, the operations, when specified, must be performed in the following order: read, punch, print. Any one function may be omitted -- that is, no file need be defined for that function. The remaining functions must still be performed in the order shown. All operations on one card must be completed before operations on the next card are begun, or there is an abnormal termination of the job. When such combined function processing is to be used, the programmer must be aware of the special considerations needed to accomplish the desired results.

COBOL handles each of the separate functions to be combined as a separate logical file. Each such logical file has its own file structure and procedural processing requirements. However, because such combined function files (also called associated files) refer to one physical unit, the user must observe certain restrictions during processing. The following sections explain the programming requirements for combined function processing in DOS American National Standard COBOL.

I -- ENVIRONMENT DIVISION CONSIDERATIONS

For combined function processing there are special considerations for the SPECIAL-NAMES Paragraph, and for the SELECT, ASSIGN, and RESERVE Clauses.

SPECIAL-NAMES Paragraph

If stacker selection of punched output is desired, mnemonic-names for the purpose can be specified in the SPECIAL-NAMES Paragraph. The mnemonic-names may be equated with the following function-names:

<u>Function Name</u>	<u>Meaning</u>
S01	Stacker 1 (for 2560, 3525)
S02	Stacker 2 (for 2560, 3525)
S03	Stacker 3 (for 2560)
S04	Stacker 4 (for 2560)
S05	Stacker 5 (for 2560)

For the 3525 device, if line control of multiline printed output is desired, mnemonic-names may be equated with the following function-names:

<u>Function Name</u>	<u>Meaning</u>
C02	Line 3
C03	Line 5
C04	Line 7
...	...
C12	Line 23

(Note that these function-names are not valid for 2-line 3525 print files, or for the 2560 print features.)

SELECT Clause

For each of the functions (reading, punching, printing) to be combined, a unique file-name must be defined.

ASSIGN Clause

For combined function card files, the ASSIGN clause has the following formats:

$$\begin{array}{l}
 \text{SYSnnn-UR-3525} \left\{ \begin{array}{l} \text{R} \\ \text{P} \\ \text{W} \\ \text{M} \end{array} \right\} - \left\{ \begin{array}{l} \text{X[R]} \\ \text{Y[R]} \\ \text{V[R]} \\ \text{Z} \end{array} \right\} [-\text{name}] \\
 \\
 \text{SYSnnn-UR-2560} \left\{ \begin{array}{l} \text{R} \\ \text{P} \\ \text{W} \end{array} \right\} - \left\{ \begin{array}{l} \text{V} \\ \text{X} \\ \text{Y} \\ \text{Z} \end{array} \right\} \left\{ \begin{array}{l} \text{[P]} \\ \text{S} \end{array} \right\} [-\text{name}]
 \end{array}$$

where nnn is a 3-digit number between 000 and 221, inclusive. Each of the associated logical files must be specified with the same SYSnnn field. (The field represents the symbolic unit to which the logical file is assigned.) The name field has the same meaning it has for other files.

The device and organization fields are interdependent. The following entries are valid:

<u>Device</u>	<u>Organization</u>	<u>Meaning</u>
3525R (reader)	V[R]	read/print associated file
	X[R]	read/punch/print associated file
	Y[R]	read/punch associated file
<u>Note:</u> The optional R code in the organization field specifies RCE (Read Column Eliminate) card reading.		
2560R (reader)	V[P]	read/print associated file, primary input hopper
	VS	read/print associated file, secondary input hopper
	X[P]	read/punch/print associated file, primary input hopper

<u>Device</u>	<u>Organization</u>	<u>Meaning</u>
2560R (reader) (continued)	XS	read/punch/print associated file, secondary input hopper
	Y[P]	read/punch associated file, primary input hopper
	YS	read/punch associated file, secondary input hopper
3525P (punch)	X	read/punch/print associated file
	Y	read/punch associated file
	Z	punch/print associated file
2560P (punch)	X[P]	read/punch/print associated file, primary input hopper
	XS	read/punch/print associated file, secondary input hopper
	Y[P]	read/punch associated file, primary input hopper
	YS	read/punch associated file, secondary input hopper
	Z[P]	punch/print associated file, primary input hopper
3525W (2-line printer)	ZS	punch/print associated file, secondary input hopper
	V	read/print associated file
	X	read/punch/print associated file
2560W (1 to 6 line printer)	Z	punch/print associated file
	V[P]	read/print associated file, primary input hopper
	VS	read/print associated file, secondary input hopper
	X[P]	read/punch/print associated file, primary input hopper
	XS	read/punch/print associated file, secondary input hopper
3525M (multiline printer)	Z[P]	punch/print associated file, primary input hopper
	ZS	punch/print associated file, secondary input hopper
	V	read/print associated file
	X	read/punch/print associated file
	Z	punch/print associated file

Note: For the 2560 device, all input hopper specifications in one combined function file structure must be identical.

RESERVE Clause

For a combined function structure, the files assigned to the read function and the punch function must each specify RESERVE NO ALTERNATE AREA(S).

A file assigned to the print function may specify either NO or 1 ALTERNATE AREA(S).

II -- DATA DIVISION CONSIDERATIONS

For each logical file defined in the Environment Division for the combined function structure, there must be a corresponding FD entry and 01 record description entry in the File Section of the Data Division.

III -- PROCEDURE DIVISION CONSIDERATIONS

When combined function processing is to be used, input/output operations must proceed in a specified order in the Procedure Division. The card passes first through the reading station, next through the punching station, and last through the printing station. Therefore, the following combined functions may be specified, in the order shown:

<u>Functions to be Combined</u>	<u>Order of Operations</u>	<u>Associated COBOL Statement</u>
read/punch/print	read punch [print]	READ ... AT END WRITE ... [ADVANCING/POSITIONING] WRITE (2560) WRITE ... [AFTER ADVANCING/POSITIONING] (3525)
read/punch	read punch	READ ... AT END WRITE ... [ADVANCING/POSITIONING]
read/print	read [print]	READ ... AT END WRITE (2560) WRITE ... [AFTER ADVANCING/POSITIONING] (3525)
punch/print	punch [print]	WRITE ... [ADVANCING/POSITIONING] WRITE (2560) WRITE ... [AFTER ADVANCING/POSITIONING] (3525)

All operations on one card must be completed before the next card is obtained, or there is an abnormal termination of the job.

The following Procedure Division considerations in the COBOL source program apply:

OPEN Statement

Combined function files may be opened in any sequence. The read function file must be opened INPUT; the punch and print function files must be opened OUTPUT. All files must be opened before processing begins; if they are not, the job is terminated.

READ Statement

For combined function files, the READ statement, if the function is specified, must be the first input/output operation specified. A second READ statement must not be issued before all necessary combined function operations for the same card have been completed, or abnormal termination of the job results.

WRITE Statement -- Punch Function Files

When the punch function is used, then after the READ statement is issued, the next input/output operation must be a WRITE statement for the punch function file.

If the user wishes to punch additional data into some of the cards and not into others, he must issue a dummy WRITE statement for the null cards, first filling the output area with SPACES.

If stacker selection for the punch function file is desired, the user can specify S01 through S05 as function-names in the SPECIAL-NAMES Paragraph. He can then issue WRITE ADVANCING statements using the associated mnemonic-names. Alternatively, if he specifies WRITE AFTER POSITIONING, he must use the identifier-2 option; the values placed in identifier-2 before the statement is issued are shown in Figure 58. Stacker selection may be specified only for the punch function file.

Value in identifier-2	Meaning	Valid for
V	Stacker 1	2560, 3525
W	Stacker 2	2560, 3525
X	Stacker 3	2560
Y	Stacker 4	2560
Z	Stacker 5	2560

Figure 58. Identifier-2 Stacker Values for WRITE AFTER POSITIONING

WRITE Statement -- Print Function Files

After the punch function operations (if specified) are completed, the user can issue WRITE statement(s) for the print function file.

If the user wishes to print additional data on some of the data cards and not on others, he may omit the WRITE statement for the null cards.

Any attempt to write beyond the limits of the card results in abnormal termination of the job; thus, the END-OF-PAGE option may not be specified.

2560 DEVICE: One WRITE statement may be issued for each card. The print feature allows a maximum of 64 characters per line and 6 lines per card; thus the maximum logical record size is 384 characters. Line control may not be specified (that is no WRITE ADVANCING/POSITIONING statements are allowed).

3525 DEVICE: Depending on the capabilities of the specific model in use, the print file may be either a 2-line print file or a multi-line print file. Up to 64 characters may be printed on each line.

For a 2-line print file, the lines are printed on line 1 (top edge of card) and line 3 (between rows 11 and 12). Line control may not be specified. Automatic spacing is provided.

For a multi-line print file up to 25 lines of characters may be printed. Line control may be specified. If line control is not specified, automatic spacing is provided.

Line control is specified by issuing WRITE AFTER ADVANCING statements, or WRITE AFTER POSITIONING statements for the print function file. If line control is used for one such statement, it must be used for all other WRITE statements issued to the file. The maximum number of printable characters, including any SPACE characters, is 64. The first character of the record defined must be reserved by the programmer for the line control character; therefore, the record may be defined as containing up to 65

characters. Such WRITE statements must not specify space suppression.

Identifier and integer have the same meanings they have for other WRITE AFTER ADVANCING or WRITE AFTER POSITIONING statements. However, such WRITE statements must not increase the line position on the card beyond the card limits, or abnormal termination results.

The mnemonic-name of the WRITE AFTER ADVANCING statement may also be specified. In the SPECIAL-NAMES Paragraph, the following function-names may be associated with the mnemonic-names:

<u>Function Name</u>	<u>Meaning</u>
C02	Line 3
C03	Line 5
C04	Line 7
...	...
C12	Line 23

(See also "System/370 Card Devices" in the Procedure Division WRITE Statement documentation.)

CLOSE Statement

When processing is completed, a CLOSE statement must be issued for each of the combined function files. After a CLOSE statement has been issued for any one of the functions, an attempt to perform processing for any of the functions results in abnormal termination.

IBM AMERICAN NATIONAL STANDARD COBOL GLOSSARY

ACCESS: The manner in which files are referenced by the computer. Access can be sequential (records are referred to one after another in the order in which they appear on the file), or it can be random (the individual records can be referred to in a nonsequential manner).

Actual Decimal Point: The physical representation, using either of the decimal point characters (. or ,), of the decimal point position in a data item. When specified, it will appear in a printed report, and it requires an actual space in storage.

ACTUAL KEY: A key which can be directly used by the system to locate a logical record on a mass storage device.

Alphabetic Character: A character which is one of the 26 characters of the alphabet, or a space. In COBOL, the term does not include any other characters.

Alphanumeric Character: Any character in the computer's character set.

Alphanumeric Edited Character: A character within an alphanumeric character string which contains at least one B or 0.

Arithmetic Expression: A statement containing any combination of data-names, numeric literals, and figurative constants, joined together by one or more arithmetic operators in such a way that the statement as a whole can be reduced to a single numeric value.

Arithmetic Operator: A symbol (single character or two-character set) which directs the system to perform an arithmetic operation. The following list shows arithmetic operators:

<u>Meaning</u>	<u>Symbol</u>
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

Assumed Decimal Point: A decimal point position which does not involve the existence of an actual character in a data item. It does not occupy an actual space in storage, but is used by the compiler to align a value properly for calculation.

BLOCK: In COBOL, a group of characters or records which is treated as an entity when moved into or out of the computer. The term is synonymous with the term Physical Record.

Buffer: A portion of main storage into which data is read or from which it is written.

Byte: A sequence of eight adjacent binary bits. When properly aligned, two bytes form a halfword, four bytes a fullword, and eight bytes a doubleword.

Channel: A device that directs the flow of information between the computer main storage and the input/output devices.

Character: One of a set of indivisible symbols that can be arranged in sequences to express information. These symbols include the letters A through Z, the decimal digits 0 through 9, punctuation symbols, and any other symbols which will be accepted by the data-processing system.

Character Set

Character Set: All the valid COBOL characters. The complete set of 51 characters is listed in "Language Considerations."

Character String: A connected sequence of characters. All COBOL characters are valid.

Checkpoint: A reference point in a program at which information about the contents of core storage can be recorded so that, if necessary, the program can be restarted at an intermediate point.

Class Condition: A statement that the content of an item is wholly alphabetic or wholly numeric. It may be true or false.

Clause: A set of consecutive COBOL words whose purpose is to specify an attribute of an entry. There are three types of clauses: data, environment, and file.

COBOL Character: Any of the 51 valid characters (see CHARACTER) in the COBOL character set. The complete set is listed in "Language Considerations."

Collating Sequence: The arrangement of all valid characters in the order of their relative precedence. The collating sequence of a computer is part of the computer design -- each acceptable character has a predetermined place in the sequence. A collating sequence is used primarily in comparison operations.

COLUMN Clause: A COBOL clause used to identify a specific position within a report line.

Comment: An annotation in the Identification Division or Procedure Division of a COBOL source program. A comment is ignored by the compiler. As an IBM extension, comments may be included at any point in a COBOL source program.

Compile Time: The time during which a COBOL source program is translated by the COBOL compiler into a machine language object program.

Compiler: A program which translates a program written in a higher level language into a machine language object program.

Compiler Directing Statement: A COBOL statement which causes the compiler to take a specific action at compile time, rather than the object program to take a particular action at execution time.

Compound Condition: A statement that tests two or more relational expressions. It may be true or false.

Condition:

- One of a set of specified values a data item can assume.
- A simple conditional expression: relation condition, class condition, condition-name condition, sign condition, switch-status condition, NOT condition.

Conditional Statement: A statement which specifies that the truth value of a condition is to be determined, and that the subsequent action of the object program is dependent on this truth value.

Conditional Variable: A data item that can assume more than one value; one or more of the values it assumes has a condition-name assigned to it.

Condition Name: The name assigned to a specific value, set of values, or range of values, that a data item may assume.

Condition-name Condition: A statement that the value of a conditional variable is one of a set (or range) of values of a data item identified by a condition-name. The statement may be true or false.

CONFIGURATION SECTION: A section of the Environment Division of the COBOL program. It describes the overall specifications of computers.

Connective: A word or a punctuation character that does one of the following:

- Associates a data-name or paragraph-name with its qualifier
- Links two or more operands in a series
- Forms a conditional expression

CONSOLE: A COBOL mnemonic-name associated with the console typewriter.

Contiguous Items: Consecutive elementary or group items in the Data Division that have a definite relationship with each other.

Control Break: A recognition of a change in the contents of a control data item that governs a hierarchy.

Control Bytes: Bytes associated with a physical record that serve to identify the record and indicate its length, blocking factor, etc.

Control Data Item: A data item that is tested each time a report line is to be printed. If the value of the data item has changed, a control break occurs and special actions are performed before the line is printed.

CONTROL FOOTING: A report group that occurs at the end of the control group of which it is a member.

Control Group: An integral set of related data that is specifically associated with a control data item .

CONTROL HEADING: A report group that occurs at the beginning of the control group of which it is a member.

Control Hierarchy: A designated order of specific control data items. The highest level is the final control; the lowest level is the minor control.

Core Storage: Storage within the central processing unit of the computer, so called because this storage exists in the form of magnetic cores.

Cylinder Index: A higher level index, always present in indexed data organization. Its entries point to track indexes.

Data Description Entry: An entry in the Data Division that is used to describe the characteristics of a data item. It consists of a level number, followed by an optional data-name, followed by data clauses that fully describe the format the data will take. An elementary data description entry (or item) cannot logically be subdivided further. A group data description entry (or item) is made up of a number of related group and/or elementary items.

DATA DIVISION: One of the four main component parts of a COBOL program. The Data Division describes the files to be used in the program and the records contained within the files. It also describes any internal Working-Storage records that will be needed (see "Data Division" for full details).

Data Item

Data Item: A unit of recorded information that can be identified by a symbolic name or by a combination of names and subscripts. Elementary data items cannot logically be subdivided. A group data item is made up of logically related group and/or elementary items and can be a logical group within a record or can itself be a complete record.

Data-name: A name assigned by the programmer to a data item in a COBOL program. It must contain at least one alphabetic character.

DECLARATIVES: A set of one or more compiler-directing sections written at the beginning of the Procedure Division of a COBOL program. The first section is preceded by the header DECLARATIVES. The last section is followed by the header END DECLARATIVES. There are three options:

1. Input/output label handling
2. Input/output error-checking procedures
3. Report Writing procedures

Each has its standard format (see "Procedure Division").

Device-number: The reference number assigned to any external device.

Digit: Any of the numerals from 0 through 9. In COBOL, the term is not used in reference to any other symbol.

DIVISION: One of the four major portions of a COBOL program:

- IDENTIFICATION DIVISION, which names the program.
- ENVIRONMENT DIVISION, which indicates the machine equipment and equipment features to be used in the program.
- DATA DIVISION, which defines the nature and characteristics of data to be processed.
- PROCEDURE DIVISION, which consists of statements directing the processing of data in a specified manner at execution time.

Division Header: The COBOL words that indicate the beginning of a particular division of a COBOL program. The four division headers are:

- IDENTIFICATION DIVISION.
- ENVIRONMENT DIVISION.
- DATA DIVISION.
- PROCEDURE DIVISION.

Division-name: The name of one of the four divisions of a COBOL program.

EBCDIC Character: Any one of the symbols included in the eight-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set. All 51 COBOL characters are included.

Editing Character: A single character or a fixed two-character combination used to create proper formats for output reports (see "Language Considerations" for a complete list of editing characters).

Elementary Item: A data item that cannot logically be subdivided.

Entry: Any consecutive set of descriptive clauses terminated by a period, written in the Identification, Environment, or Procedure Divisions of a COBOL program.

Entry-name: A programmer-specified name that establishes an entry point into a COBOL subprogram.

ENVIRONMENT DIVISION: One of the four main component parts of a COBOL program. The Environment Division describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored (see "Environment Division" for full details).

Execution Time: The time at which an object program actually performs the instructions coded in the Procedure Division, using the actual data provided.

Exponent: A number, indicating how many times another number (the base) is to be repeated as a factor. Positive exponents denote multiplication, negative exponents denote division, fractional exponents denote a root of a quantity. In COBOL, exponentiation is indicated with the symbol ** followed by the exponent.

F-mode Records: Records of a fixed length, each of which is wholly contained within a block. Blocks may contain more than one record.

Figurative Constant: A reserved word that represents a numeric value, a character, or a string of repeated values or characters. The word can be written in a COBOL program to represent the values or characters without being defined in the Data Division (see "Language Considerations" for a complete list).

FILE-CONTROL: The name and Header of an Environment Division paragraph in which the data files for a given source program are named and assigned to specific input/output devices.

File Description: An entry in the File Section of the Data Division that provides information about the identification and physical structure of a file.

File-name: A name assigned to a set of input data or output data. A file-name must include at least one alphabetic character.

FILE SECTION: A section of the Data Division that contains descriptions of all externally stored data (or files) used in a program. Such information is given in one or more file description entries.

Floating-Point Literal: A numeric literal whose value is expressed in floating-point notation -- that is, as a decimal number followed by an exponent which indicates the actual placement of the decimal point.

Function-name: A name, supplied by IBM, that identifies system logical units, printer and card punch control characters, and report codes. When a function-name is associated with a mnemonic-name in the Environment Division, the mnemonic-name can then be substituted in any format in which substitution is valid.

Group Item: A data item made up of a series of logically related elementary items. It can be part of a record or a complete record.

Header Label: A record that identifies the beginning of a physical file or a volume.

High-Order : The leftmost position in a string of characters.

IDENTIFICATION DIVISION: One of the four main component parts of a COBOL program. The Identification Division identifies the source program and the object program and, in addition, may include such documentation as the author's name, the installation where written, date written, etc. (see "Identification Division" for full details).

Identifier: A data-name, unique in itself, or made unique by the syntactically correct combination of qualifiers, subscripts, and/or indexes.

Imperative-Statement: A statement consisting of an imperative verb and its operands, which specifies that an action be taken, unconditionally. An imperative-statement may consist of a series of imperative-statements.

Index: A computer storage position or register, the contents of which identify a particular element in a table.

Index Data Item: A data item in which the contents of an index can be stored without conversion to subscript form.

Index-name: A name, given by the programmer, for an index of a specific table. An index-name must contain at least one alphabetic character. It is one word (4 bytes) in length.

Indexed Data-name: A data-name identifier which is subscripted with one or more index-names.

INPUT-OUTPUT SECTION: In the Environment Division, the section that names the files and external media needed by an object program. It also provides information required for the transmission and handling of data during the execution of an object program.

INPUT PROCEDURE: A set of statements that is executed each time a record is released to the sort file. Input procedures are optional; whether they are used or not depends upon the logic of the program.

Integer: A numeric data item or literal that does not include any character positions to the right of the decimal point, actual or assumed. Where the term "integer" appears in formats, "integer" must not be a numeric data item.

INVALID KEY Condition: A condition that may arise at execution time in which the value of a specific key associated with a mass storage file does not result in a correct reference to the file (see the READ, REWRITE, START, and WRITE statements for the specific error conditions involved).

I-O-CONTROL: The name, and the header, for an Environment Division paragraph in which object program requirements for specific input/output techniques are specified. These techniques include rerun checkpoints, sharing of same areas by several data files, and multiple file storage on a single tape device.

KEY: One or more data items, the contents of which identify the type or the location of a record, or the ordering of data.

Key Word: A reserved word whose employment is essential to the meaning and structure of a COBOL statement. In this manual, key words are indicated in the formats of statements by underscoring. Key words are included in the reserved word list.

Level Indicator: Two alphabetic characters that identify a specific type of file, or the highest position in a hierarchy. The level indicators are: FD, SD, RD.

Level Number: A numeric character or two-character set that identifies the properties of a data description entry. Level numbers 01 through 49 define group items, the highest level being identified as 01, and the subordinate data items within the hierarchy being identified with level numbers 02 through 49. Level numbers 66, 77, and 88 identify special properties of a data description entry in the Data Division.

Library-name: The name of a member of a data set containing COBOL entries, used with the COPY and BASIS statements.

LINKAGE SECTION: A section of the Data Division that describes data made available from another program.

Literal: A character string whose value is implicit in the characters themselves. The numeric literal 7 expresses the value 7, and the nonnumeric literal "CHARACTERS" expresses the value CHARACTERS.

Logical Operator: A COBOL word that defines the logical connections between relational operators. The three logical operators and their meanings are:

OR (logical inclusive -- either or both)

AND (logical connective -- both)

NOT (logical negation)

(See "Procedure Division" for a more detailed explanation.)

Logical Record: The most inclusive data item, identified by a level-01 entry. It consists of one or more related data items.

Low-Order: The rightmost position in a string of characters.

Main Program: The highest level COBOL program involved in a step. (Programs written in other languages that follow COBOL linkage conventions are considered COBOL programs in this sense.)

Mantissa: The decimal part of a logarithm. Therefore, the part of a floating-point number that is expressed as a decimal fraction.

Mass Storage: A storage medium -- disk, drum, or data cell -- in which data can be collected and maintained in a sequential, direct, or indexed organization.

Mass Storage File: A collection of records assigned to a mass storage device.

Mass Storage File Segment: A part of a mass storage file whose beginning and end are defined by the FILE-LIMIT clause in the Environment Division.

Master Index: The highest level index, which is optional, in the indexed data organization.

Mnemonic-name: A programmer-supplied word associated with a specific function-name in the Environment Division. It then may be written in place of the function-name in any format where such a substitution is valid.

MODE: The manner in which records of a file are accessed or processed.

Name

Name: A word composed of not more than 30 characters, which defines a COBOL operand (see "Language Considerations" for a more complete discussion).

Noncontiguous Item: A data item in the Working-Storage Section of the Data Division which bears no relationship with other data items.

Nonnumeric Literal: A character string bounded by quotation marks, which means literally itself. For example, "CHARACTER" is the literal for and means CHARACTER. The string of characters may include any characters in the computer's set, with the exception of the quotation mark. Characters that are not COBOL characters may be included.

Numeric Character: A character that belongs to one of the set of digits 0 through 9.

Numeric Edited Character: A numeric character which is in such a form that it may be used in a printed output. It may consist of external decimal digits 0 through 9, the decimal point, commas, the dollar sign, etc., as the programmer wishes (see "Data Division" for a fuller explanation).

Numeric Item: An item whose description restricts its contents to a value represented by characters from the digits 0 through 9. The item may also contain a leading or trailing operational sign represented either as an overpunch or as a separate character.

Numeric Literal: A numeric character or string of characters whose value is implicit in the characters themselves. Thus, 777 is the literal as well as the value of the number 777.

OBJECT-COMPUTER: The name of an Environment Division paragraph in which the computer upon which the object program will be run is described.

Object Program: The set of machine language instructions that is the output from the compilation of a COBOL source program. The actual processing of data is done by the object program.

Object Time: The time during which an object program is executed.

Operand: The "object" of a verb or an operator. That is, the data or equipment governed or directed by a verb or operator.

Operational Sign: An algebraic sign associated with a numeric data item, which indicates whether the item is positive or negative.

Optional Word: A reserved word included in a specific format only to improve the readability of a COBOL statement. If the programmer wishes, optional words may be omitted.

OUTPUT PROCEDURE: A set of programmer-defined statements that is executed each time a sorted record is returned from the sort file. Output procedures are optional; whether they are used or not depends upon the logic of the program.

Overlay: The technique of repeatedly using the same areas of internal storage during different stages in processing a problem.

PAGE: A physical separation of continuous data in a report. The separation is based on internal requirements and/or the physical characteristics of the reporting medium.

PAGE FOOTING: A report group at the end of a report page which is printed before a page control break is executed.

PAGE HEADING: A report group printed at the beginning of a report page, after a page control break is executed.

Paragraph: A set of one or more COBOL sentences, making up a logical processing entity, and preceded by a paragraph-name or a paragraph header.

Paragraph Header: A word followed by a period that identifies and precedes all paragraphs in the Identification Division and Environment Division.

Paragraph-name: A programmer-defined word that identifies and precedes a paragraph.

Parameter: A variable that is given a specific value for a specific purpose or process. In COBOL, parameters are most often used to pass data values between calling and called programs.

Physical Record: A physical unit of data, synonymous with a block. It can be composed of a portion of one logical record, of one complete logical record, or of a group of logical records.

Print Group: An integral set of related data within a report.

Priority-Number: A number, ranging in value from 0 to 99, which classifies source program sections in the Procedure Division (see "Segmentation" for more information).

Procedure: One or more logically connected paragraphs or sections within the Procedure Division, which direct the computer to perform some action or series of related actions.

PROCEDURE DIVISION: One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative-statements, conditional statements, paragraphs, procedures, and sections (see "Procedure Division" for full details).

Procedure-name: A word that precedes and identifies a procedure, used by the programmer to transfer control from one point of the program to another.

Process: Any operation or combination of operations on data.

Program-name: A word in the Identification Division that identifies a COBOL source program.

Punctuation Character: A comma, semicolon, period, quotation mark, left or right parenthesis, or a space.

Qualifier: A group data-name that is used to reference a non-unique data-name at a lower level in the same hierarchy, or a section-name that is used to reference a non-unique paragraph. In this way, the data-name or the paragraph-name can be made unique.

Random Access: An access mode in which specific logical records are obtained from, or placed into, a mass storage file in a nonsequential manner.

RECORD: A set of one or more related data items grouped for handling either internally or by the input/output systems (see "Logical Record").

Record Description: The total set of data description entries associated with a particular logical record.

Record-name: A data-name that identifies a logical record.

Reel: A module of external storage associated with a tape device.

Relation Character

Relation Character: A character that expresses a relationship between two operands. The following are COBOL relation characters:

<u>Character</u>	<u>Meaning</u>
>	Greater than
<	Less than
=	Equal to

Relation Condition: A statement that the value of an arithmetic expression or data item has a specific relationship to another arithmetic expression or data item. The statement may be true or false.

Relational Operator: A reserved word, or a group of reserved words, or a group of reserved words and relation characters. A relational operator plus programmer-defined operands make up a relational expression. A complete listing is given in "Procedure Division."

REPORT: A presentation of a set of processed data described in a Report File.

Report Description Entry: An entry in the Report Section of the Data Division that names and describes the format of a report to be produced.

Report File: A collection of records, produced by the Report Writer, that can be used to print a report in the desired format.

REPORT FOOTING: A report group that occurs, and is printed, only at the end of a report.

Report Group: A set of related data that makes up a logical entity in a report.

REPORT HEADING: A report group that occurs, and is printed, only at the beginning of a report.

Report Line: One row of printed characters in a report.

Report-name: A data-name that identifies a report.

REPORT SECTION: A section of the Data Division that contains one or more Report Description entries.

Reserved Word: A word used in a COBOL source program for syntactical purposes. It must not appear in a program as a user-defined operand.

Routine: A set of statements in a program that causes the computer to perform an operation or series of related operations.

Run Unit: A set of one or more object programs that function, at object time, as a unit to provide problem solutions. This compiler considers a run unit to be the highest level calling program plus all called subprograms.

S-mode Records: Records that span physical blocks. Records may be fixed or variable in length. Blocks may contain one or more segments. A segment may contain one record or a portion of a record. Each segment contains a segment-length field and a control field indicating whether or not it is the first and/or last or an intermediate segment of the record. Each block contains a block-length field.

SECTION: A logically related sequence of one or more paragraphs. A section must always be named.

Section Header: A combination of words that precedes and identifies each section in the Environment, Data, and Procedure Divisions.

Section-name: A word specified by the programmer that precedes and identifies a section in the Procedure Division.

Sentence: A sequence of one or more statements, the last ending with a period followed by a space.

Separator: An optional word or character that improves readability.

Sequential Access: An access mode in which logical records are obtained from, or placed into, a file in such a way that each successive access to the file refers to the next subsequent logical record in the file. The order of the records is established by the programmer when creating the file.

Sequential Processing: The processing of logical records in the order in which records are accessed.

Sign Condition: A statement that the algebraic value of a data item is less than, equal to, or greater than zero. It may be true or false.

Simple Condition: An expression that can have two values, and causes the object program to select between alternate paths of control, depending on the value found. The expression can be either true or false.

Slack Bytes: Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

Sort File: A collection of records that is sorted by a SORT statement. The sort file is created and used only while the sort function is operative.

Sort File Description Entry: An entry in the File Section of the Data Division that names and describes a collection of records that is used in a SORT statement.

Sort-file-name: A data-name that identifies a Sort File.

Sort-key: The field within a record on which a file is sorted.

Sort-work-file: A collection of records involved in the sorting operation as this collection exists on intermediate device(s).

SOURCE-COMPUTER: The name of an Environment Division paragraph. In it, the computer upon which the source program will be compiled is described.

Source Program: A problem-solving program written in COBOL.

Special Character: A character that is neither numeric nor alphabetic. Special characters in COBOL include the space (), the period(.), as well as the following:

+ - * / = \$, ; ") (

SPECIAL-NAMES: The name of an Environment Division paragraph, and the paragraph itself, in which names supplied by IBM are related to mnemonic-names specified by the programmer. In addition, this paragraph can be used to exchange the functions of the comma and the period, or to specify a substitution character for the currency sign, in the PICTURE string.

Special Register

Special Register: Compiler-generated storage areas primarily used to store information produced with the use of specific COBOL features. The special registers are: TALLY, LINE-COUNTER, PAGE-COUNTER, CURRENT-DATE, TIME-OF-DAY, COM-REG, SORT-RETURN, SORT-FILE-SIZE, SORT-CORE-SIZE, SORT-MODE-SIZE, and NSTD-REELS.

Standard Data Format: The concept of actual physical or logical record size in storage. The length in the Standard Data Format is expressed in the number of bytes a record occupies and not necessarily the number of characters, since some characters take up one full byte of storage and others take up less.

Statement: A syntactically valid combination of words and symbols written in the Procedure Division. A statement combines COBOL reserved words and programmer-defined operands.

Subject of entry: A data-name or reserved word that appears immediately after a level indicator or level number in a Data Division entry. It serves to reference the entry.

Subprogram: A COBOL program that is invoked by another COBOL program. (Programs written in other languages that follow COBOL linkage conventions are COBOL programs in this sense.)

Subscript: An integer or a variable whose value references a particular element in a table.

Switch-status Condition: A statement that an UPSI switch has been set to an ON or OFF condition. The statement may be true or false.

SYSIPT: The system input device.

SYSLST: The system output device.

SYSPCH: The system punch device.

SYSPUNCH: An alternate name for the system punch device.

System-name: A name, specified by IBM, that identifies any particular external device used with the computer, and characteristics of files contained within it.

Table: A collection and arrangement of data in a fixed form for ready reference. Such a collection follows some logical order, expressing particular values (functions) corresponding to other values (arguments) by which they are referenced.

Table Element: A data item that belongs to the set of repeated items comprising a table.

Test Condition: A statement that, taken as a whole, may be either true or false, depending on the circumstances existing at the time the expression is evaluated.

Trailer Label: A record that identifies the ending of a physical file or of a volume.

U-mode Records: Records of undefined length. They may be fixed or variable in length; there is only one record per block.

Unary Operator: An arithmetic operator (+ or -) that can precede a single variable, a literal, or a left parenthesis in an arithmetic expression. The plus sign multiplies the value by +1; the minus sign multiplies the value by -1.

UNIT: A module of external storage. Its dimensions are determined by IBM.

V-mode Records: Records of variable length, each of which is wholly contained within a block. Blocks may contain more than one record. Each record contains a record length field, and each block contains a block length field.

Variable: A data item whose value may be changed during execution of the object program.

Verb: A COBOL reserved word that expresses an action to be taken by a COBOL compiler or an object program.

Volume: A module of external storage. For tape devices it is a reel; for mass storage devices it is a unit.

Volume Switch Procedures: Standard procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

Word:

1. In COBOL: A string of not more than 30 characters, chosen from the following: the letters A through Z, the digits 0 through 9, and the hyphen (-). The hyphen may not appear as either the first or last character.
2. In System/360: A fullword is 4 bytes of storage; a doubleword is 8 bytes of storage; a halfword is 2 bytes of storage.

Word Boundary: Any particular storage position at which data must be aligned for certain processing operations in System/360. The halfword boundary must be divisible by 2, the fullword boundary must be divisible by 4, the doubleword boundary must be divisible by 8.

WORKING-STORAGE SECTION: A section-name (and the section itself) in the Data Division. The section describes records and noncontiguous data items that are not part of external files, but are developed and processed internally. It also defines data items whose values are assigned in the source program.

(When more than one page reference is given, the major reference appears first. Entries ending in "(DOS/VS)" apply only to the DOS/VS COBOL implementation. All other entries apply both to DOS/VS COBOL and DOS Full American National Standard COBOL.)

Special Characters

- [(see braces)
 - { (see brackets)
 - (see pound sign)
 - . (see period)
 - ... (see ellipsis)
 - < in relation conditions 158,159
 - (and) in
 - arithmetic expressions 153,154
 - compound conditions 162-165
 - PICTURE clause 118-120
 - subscripting and indexing 289-291
 - + (see plus symbol)
 - \$ (see currency symbol, dollar sign)
 - * in
 - arithmetic expressions 153,154
 - intermediate results 333,334
 - (see also asterisks, used in PICTURE clause)
 - ** in
 - arithmetic expressions 153,154
 - intermediate results 333,334
 - ; in Data Division and Procedure Division entries 109,149
 - (see also semicolon)
 - (see either hyphen, or minus symbol)
 - / in
 - arithmetic expressions 153,154
 - intermediate results 333,334
 - sterling report items 327-329
 - , (see comma)
 - > in relation conditions 158,159
 - = in
 - COMPUTE statement 182
 - relation conditions 158,159
 - ' or " in nonnumeric literals 42,37
 - (see also quotation mark)
-
- A, in PICTURE clause
 - alphabetic items 120
 - alphabetic symbol 118
 - alphanumeric edited items 124
 - alphanumeric items 120
 - abbreviations
 - ADD CORRESPONDING statement 181
 - compound conditions 164,165
 - CORRESPONDING option 179
 - END-OF-PAGE option 213
 - Identification Division Header 57,46
 - JUSTIFIED clause 116
 - MOVE CORRESPONDING statement 198
 - PICTURE clause 117
 - relation conditions 164,165
 - relational operators 158
 - reserved words 40
 - SUBTRACT CORRESPONDING statement 185
 - SYNCHRONIZED clause 130
 - TYPE clause 267
 - USAGE clause 136
 - absolute
 - column number 269
 - line spacing in a report 263-265
 - values in MOVE statement 199
 - ACCEPT statement 219,220
 - ACCEPT statement enhancement (DOS/VS) xxxvi
 - access methods
 - direct files 60,61
 - indexed files 61,62
 - sequential files 60
 - ACCESS MODE and VSAM capabilities (DOS/VS) xiv
 - ACCESS MODE clause 78
 - ACCESS MODE clause, VSAM (DOS/VS) vii,v
 - acknowledgment 4
 - actual decimal point
 - description 119
 - in editing 124-128
 - ACTUAL KEY clause
 - description and format 79-81
 - READ statement 211,212
 - REWRITE statement 218,219
 - SEEK statement 210
 - WRITE statement 218
 - ACTUAL KEY clause for 3340 (DOS/VS) xxxv
 - ADD statement
 - description and formats 181,182
 - addition operator 153,154
 - addressing schemes
 - direct 59,60
 - indexed 60
 - sequential 59
 - VSAM (DOS/VS) iv
 - algebraic value in a sign condition 162
 - algorithm
 - ACTUAL KEY example 336
 - direct indexing 298
 - intermediate results 333,334
 - relative indexing 299
 - slack bytes
 - computational items 119
 - inter-record 135,136
 - intra-record 132-135
 - alignment of data items
 - comparisons 159,160
 - decimal point 119,199
 - editing 125-128
 - File Section Items 131
 - JUSTIFIED clause 116

- Linkage Section Items 131
- PICTURE clause 119,125-128
- SYNCHRONIZED clause 130,131
- USING option 229
- VALUE clause 143
- Working-Storage items 131
- ALL literal figurative constant
 - description 43
 - MOVE statement 200
 - relation condition 161
 - STOP statement 196
- alphabetic character 118
- alphabetic class test 156
- alphabetic collating sequence 244
- alphabetic data items
 - allowable symbols 120
 - in a class test 156
 - description 118
 - internal representation 120,137
 - JUSTIFIED clause 116
 - MOVE statement 199,200
 - relation condition 159-161
 - USAGE clause 136,137
 - VALUE clause 143
- alphabetized cross-reference listing (SXREF) 12
- alphanumeric character 119
- alphanumeric collating sequence 244
- alphanumeric data item
 - allowable symbols 120
 - class test 156
 - description 120
 - internal representation 120,137
 - JUSTIFIED clause 116
 - MOVE statement 199,200
 - relation condition 159-161
 - USAGE clause 136,137
 - VALUE clause 143
- alphanumeric data item (DOS/VS)
 - FILE STATUS clause, VSAM viii
 - RECORD KEY clause, VSAM vii,viii
- alphanumeric edited character 124
- alphanumeric edited item
 - allowable symbols 124
 - description 124
 - MOVE statement 199,200
 - relation condition 159-161
 - USAGE clause 136,137
 - VALUE clause 143
- alphanumeric literals (see nonnumeric literals)
- ALTER statement
 - and called programs 227
 - in debug packets 322
 - description and format 188
 - effect on GO TO statement 188
 - segmentation 312
 - sort procedure 245,246
- altering characters 201-205
- altering execution sequence 187-197
- altering usage of data items 114,115
- alternative grouping of data
 - REDEFINES clause 112-115
 - RENAMES clause 144-146
- AND logical operator
 - compound conditions 162-165
 - order of evaluation 163
- apostrophe (see quotation mark)
- APPLY clause
 - CORE-INDEX option 89
 - CYL-INDEX option 89
 - CYL-OVERFLOW option 88
 - EXTENDED-SEARCH option 87
 - MASTER-INDEX option 89
 - WRITE-ONLY option 87
 - WRITE-VERIFY option 88
- APPLY CYL-OVERFLOW clause for 3340 (DOS/VS) xxxv
- APPLY WRITE-VERIFY invalid for 3540 (DOS/VS) xxxv
- arabic numerals 122,123
- Area A and Area B in reference format 50,51
- arithmetic expressions
 - characters used 153
 - COMPUTE statement 182
 - conditions 158
 - description 153,154
 - evaluation rules 154
- arithmetic operators
 - description and list 153,39
 - order of evaluation 153
 - symbol pairs 154
- arithmetic statements
 - ADD 181,182
 - COMPUTE 182
 - CORRESPONDING option 179
 - DIVIDE 183
 - GIVING option 179
 - intermediate results 333,334
 - MULTIPLY 184
 - overlapping operands 180
 - REMAINDER option 183
 - ROUNDED option 179,180
 - SIZE ERROR option 180
 - SUBTRACT 185,186
- ascending sequence
 - ASCII character set 360
 - EBCDIC character set 159,244
 - sort 243,244
 - table handling 293,295,296
- ascending sequence, merge (DOS/VS) xxv,xxvi
- ASCII considerations 355-361
- ASSIGN clause
 - ASCII considerations 355,356
 - description and format 69-75
 - NSTD-REELS special register 70,45
 - sort
 - ASCII considerations 360,361
 - file in GIVING option 238,239
 - sort work units 239
 - system-name 70-75
 - Version 3 considerations 72-75
- ASSIGN clause (DOS/VS)
 - merge facility xxii
 - VSAM files v,vi
 - 3340 and 3540 devices xxxiv
- assumed
 - decimal point
 - description 119
 - numeric edited items 124
 - numeric items 122,123
 - sterling nonreport items 325,326
 - decimal scaling positions 119,122-124
 - pound and shilling separators 325

asterisk
 arithmetic expressions 153,154
 comments 52,234
 PICTURE clause
 check protect symbol 120
 numeric edited items 124,128
 sterling report items 327-329
 AT END phrase
 READ statement 211,212
 RETURN statement 248
 SEARCH statement 301-305
 AUTHOR paragraph 57
 automatic
 advancing of printer page 214
 end-of-volume 212
 error procedures 175,178
 label handling 171,105

 B, in PICTURE clause
 alphanumeric edited items 124
 numeric edited items 124
 space symbol 118
 sterling report items 327-329
 BASIS card 316
 BEFORE REPORTING declarative 276,277
 binary collating sequence 244
 binary data item
 in PICTURE clause 118
 description 138,139
 internal representation 140
 MOVE statement 200
 relation condition 161
 SYNCHRONIZED clause 130,131
 USAGE clause 136,138-140
 blank (see space)
 BLANK clause (see BLANK WHEN ZERO clause)
 blank figurative constant (see SPACE figurative constant)
 blank line
 definition and use 52
 for spacing reports 265,266
 blank (space) as word separator 40
 BLANK WHEN ZERO clause
 description and format 115
 with sterling report items 329
 BLOCK CONTAINS clause
 ASCII considerations 356
 description and format 100,101
 BLOCK CONTAINS clause (DOS/VS)
 for 3540 xxxv
 VSAM ix
 block-descriptor control field 103,104
 block size 100,101
 blocked records
 APPLY WRITE-ONLY clause 87
 BLOCK CONTAINS clause 100,101
 inter-record slack bytes 135,136
 recording mode 103,104
 body print group 258-260
 boundary alignment 131-136,138
 braces and brackets in formats 53
 British Standards Institution (BSI) 324
 buffer
 allocation 77
 ASCII considerations 356
 combined function processing 381
 truncation 87

 buffer allocation VSAM (DOS/VS) vi
 bypassing label processing
 LABEL RECORDS clause 105,106
 MULTIPLE FILE TAPE clause 86,87
 byte, contents of
 alphabetic and alphanumeric items 137
 binary item 138,140
 external decimal items 137,140
 external floating-point items 137,141
 internal decimal items 139,141
 internal floating-point items 138,141

 C, in sterling PICTURE 327-329
 CALL statement
 boundary alignment in 229
 description and format 226,227
 limitations with segmentation 312
 USING option 228-231
 CALL statement for 3886 OCR processing (DOS/VS) xxviii
 capitalized words in formats 53
 carriage control character
 definition 68
 System/370 card devices
 combined function processing 383,384
 WRITE statement 216,217
 WRITE statement 212-216
 categories of data (see PICTURE clause)
 changing description of data
 items 112-115,144-146
 character set
 arithmetic expressions 153,39
 ASCII 360
 COBOL 37
 EBCDIC 37
 editing 39
 punctuation 38
 relation-conditions 158,39
 words 37
 character string
 and item size 118
 NOTE statement 233,234
 PICTURE clause 117-121
 truncation 116
 check protect symbol (see asterisk)
 checkpoint
 description and format 84,85
 sort considerations 240
 class test 156
 classes of data 117
 CLOSE statement
 description and format 221-225
 OPEN REVERSED statement 207
 random file options 224,225
 sequential file options 222-224
 CLOSE statement (DOS/VS)
 VSAM xx,xxi
 3540 device xxxvi
 CODE clause 256,257,66,67
 description and format 256,257
 SPECIAL-NAMES paragraph 66,67
 coding form, COBOL 50
 collating sequence
 ASCII 360
 EBCDIC 159,244
 for sort 244

COLUMN clause 269
 combined function processing 379-384
 combined function processing, 5425
 (DOS/VS) xxxvi
 comma, exchanging with period
 international considerations 330
 SPECIAL-NAMES paragraph 66,67
 comma
 in editing 124-128
 PICTURE string 119,121
 as punctuation 38
 comment-entry
 asterisk preceding 52,233,234
 DATE-COMPILED paragraph 58
 Identification Division 57,58
 NOTE statement 233,234
 comment lines
 in every division 234,52
 in Procedure Division 233,234
 common exit point for procedures
 and PERFORM statement 190
 in program 196,197
 common processing facilities, VSAM (DOS/VS)
 current record pointer x
 INTO/FROM identifier option xi
 INVALID KEY condition xi
 status key x
 communication
 operating system
 sort special registers 249
 system special registers 44,45
 operator
 ACCEPT statement 219,220
 ASSIGN clause 69
 STOP statement 196
 sort feature 249
 subprogram 226-232
 COMP items (see binary data items)
 COMP-1 items (see short precision internal
 floating-point items)
 COMP-2 items (see long precision internal
 floating-point data items)
 COMP-3 items (see internal decimal items)
 COMP-4 items (see binary data items)
 comparison
 index data items 300,301
 index-names 300,301
 operands 158
 in relation conditions 161
 compilation of
 copied text 313-315
 debugging packet 322
 compile-time debugging packet 322
 compiler directing statements
 BASIS 316
 COPY 313-315
 DEBUG 322
 description 152
 DELETE 316,317
 EJECT 323
 ENTER 233
 INSERT 316,317
 list of 152
 NOTE 233,234
 SKIP 323
 compiler options
 current-date 44
 quotation mark 37
 sequence checking 50
 truncation of binary items 118
 compound conditions
 description 162-165
 evaluation rules 163
 implied subjects and
 relational-operators 164,165
 logical operators 162
 permissible symbol pairs 164
 SEARCH statement 305
 COMPUTATIONAL items (see binary data items)
 COMPUTATIONAL usage
 descriptions and format 138,136
 internal representation 140
 COMPUTATIONAL-1 items (see short precision
 internal floating-point items)
 COMPUTATIONAL-1 and COMPUTATIONAL-2 usage
 descriptions and format 138,136
 internal representation 141
 COMPUTATIONAL-2 items (see long precision
 internal floating-point data items)
 COMPUTATIONAL-3 items (see internal decimal
 items)
 COMPUTATIONAL-3 usage
 description and format 139,136
 internal representation 141
 COMPUTATIONAL-4 items (see binary data
 items)
 COMPUTATIONAL-4 usage
 description and format 139,136
 internal representation 140
 COMPUTE statement 182
 computer-name
 OBJECT-COMPUTER paragraph 65
 SOURCE-COMPUTER paragraph 64
 System/370 instructions 65
 computer-name (DOS/VS) ii-iv
 COM-REG special register 44
 condition-name (see level number 88 items)
 condition-name condition
 description and format 157
 and SEARCH statement 305
 conditional sentence, definition 166
 conditional statement
 definition 166,150
 IF statement 166-168
 list of 151
 ON statement 320,321
 conditional variable
 assigning values to 142-144
 condition-name condition 157
 example 144
 qualification of 48,49
 conditions
 compound conditions 162-165
 PERFORM statement
 descriptions 191-195
 formats 189
 SEARCH statement 301-305
 test conditions 155-162
 Configuration Section
 copying 313-315
 description and format 64-67
 OBJECT-COMPUTER paragraph 65
 SOURCE-COMPUTER paragraph 64
 SPECIAL-NAMES paragraph 65-67
 and System/370 instruction
 generation 65

Configuration Section (DOS/VS)
 description and format ii-iv
 OBJECT-COMPUTER paragraph ii-iv
 SOURCE-COMPUTER paragraph ii
 System/370 instruction generation ii-iv
 CONSOLE
 ACCEPT statement 219,220
 DISPLAY statement 220,221
 SPECIAL-NAMES paragraph 66
 constant
 definition 41
 figurative 43
 literal 41
 continuation area
 comment lines 52,234
 reference format 50
 continuation line 51
 continuation of
 ACCEPT operands 219,220
 comments 52,234
 DISPLAY operands 220,221
 lines 51
 nonnumeric literals 52
 NOTE statement 233,234
 numeric literals 52
 words 52
 continued line 51
 control breaks (see CONTROL clause)
 control bytes
 BLOCK CONTAINS clause 101
 inter-record slack bytes 135,136
 S-mode and V-mode records 103
 CONTROL clause 257,258
 control hierarchy 257,258
 control of sort procedures 245-247
 CONTROL report group
 CONTROL clause 257,258
 GENERATE statement 273,274
 LINE clause 263-265
 NEXT GROUP clause 265,266
 PAGE LIMIT clause 258-260
 report group description entry 261-263
 summation 272,273
 TYPE clause 267-269
 controls in report writer (see CONTROL clause)
 conversion of data (see data conversion)
 COPY statement 313-315
 copying
 entire program 316,317
 part of a program 313-315
 CORE-INDEX option of the APPLY clause 89
 core storage for sort
 SAME clause 240,241
 SORT-CORE-SIZE special register 249
 CORRESPONDING option
 arithmetic statements
 ADD 181
 description 179
 SUBTRACT 185,186
 MOVE statement 198,179
 counter updating 272,273
 CR, in PICTURE clause
 description 120,121
 numeric edited items 124,128
 sterling report items 327-329
 creating files
 direct 60,61
 indexed 61,62
 sample programs 336-338
 standard sequential 60
 (see also output files)
 creating labels 170-175
 credit symbol (see CR in PICTURE clause)
 cross-footing 272
 CSP function-name defined 66
 CURRENCY-SIGN clause
 description and format 66,67
 international considerations 330
 restriction 67
 currency symbol in PICTURE clause
 (see also CURRENCY SIGN clause)
 dollar sign
 description 120,121
 numeric edited items 124-128
 pound sign 327-329
 CURRENT-DATE special register 44
 current record pointer, VSAM (DOS/VS) ix
 cylinder overflow 88
 CYL-INDEX option, APPLY clause 89
 CYL-OVERFLOW option, APPLY clause 88
 C01 through C12 function-names defined 66

 D, in sterling PICTURE clause 325-329
 data, categories of (see PICTURE clause)
 data conversion
 DISPLAY statement 220,221
 EXAMINE statement 201,202
 first character of program-name 58
 GIVING option 179
 MOVE statement 199
 TRANSFORM statement 203-205
 data description clauses
 BLANK WHEN ZERO 115
 data-name 111
 FILLER 111
 JUSTIFIED 116
 OCCURS 292-299
 PICTURE 116-128
 REDEFINES 112-115
 RENAMES 144-146
 SYNCHRONIZED 130,131
 USAGE 136-141
 VALUE 142-144
 data description entry
 (see also "data description clauses")
 ASCII considerations 357
 definition 108
 general formats 108,109
 indentation 97
 maximum length 108
 data description entry, VSAM (DOS/VS) ix
 data description terms 108
 Data Division
 ASCII considerations 357
 description 91-146
 organization 94
 report writer considerations
 File Section 254,255
 Report Section 256-273
 sort considerations 241,242
 structure 94
 table handling considerations 292-299
 Data Division, VSAM (DOS/VS) ix

- data item
 - assigning a value to 142-144
 - definition 108
 - maximum length 108
 - overlapping
 - arithmetic statements 180
 - MOVE statement 199
 - TRANSFORM statement 203
- data item alignment (see "alignment of data items")
- data item description entry
 - description 108
 - Linkage Section 98,99
 - Working-Storage Section 98
- data management techniques 59-62
- data manipulation statements
 - EXAMINE 201,202
 - MOVE 198-200
 - TRANSFORM 203-205
- data-name
 - definition 41
 - qualification of 48,49
 - in reference format 51
- data-name clause 111
- data organization
 - direct 59-60
 - indexed 60
 - sequential 59
 - specification of 71-75
- data organization, VSAM (DOS/VS) iv
- DATA RECORDS clause
 - description and format 106,107
 - report writer considerations 255
 - sort considerations 241,242
- DATA RECORDS clause, VSAM (DOS/VS) ix
- data reference methods 48,49
- data sets for symbolic debugging 364,365
- data transformation 203-205
- DATE-COMPILED paragraph 58
- DATE-WRITTEN paragraph 57
- DB, in PICTURE clause
 - description 120,121
 - numeric edited items 124-128
 - sterling report items 327-329
- debit symbol (see DB, in PICTURE clause)
- DEBUG card 322
- debugging, symbolic 363-377
- debugging language
 - output 318
 - packet 322
 - statements
 - DEBUG card 322
 - EXHIBIT 318-320
 - ON 320,321
 - READY/RESET TRACE 318
- decimal point (see period, used in a PICTURE clause)
- decimal point alignment
 - MOVE statement 199
 - period insertion character 119,124
 - rounding 179,180
 - size error 180
- DECIMAL-POINT IS COMMA clause
 - description and format 66,67
 - international considerations 330
- declaratives
 - error processing 175-178
 - EXIT statement in 177,178
- label handling 170-175
- report writer 276,277
- section
 - description and format 169,150
- USE sentence 169
- defaults
 - ACCESS MODE clause 78
 - APPLY CYL/MASTER-INDEX clause 89
 - BLOCK CONTAINS clause 101
 - cylinder overflow area 88
 - name-field in system-name 71
 - page format in Report Writer 260
 - printer spacing 214
 - priority number 311
 - quotation mark character 37
 - record sizes in DISPLAY 220
 - recording mode 103,104
 - segment limit 311
 - sequence checking 50
 - truncation of binary items 118
 - USAGE clause 137
- definitions of terms 385-397
- DELETE card for copying 316-317
- DELETE statement, VSAM (DOS/VS)
 - description and format xx,xxi
 - processing capabilities xiv
- DEPENDING ON option
 - GO TO statement
 - description and format 187
 - maximum number of operands 187
- OCCURS clause
 - description and formats 292-299
 - logical record size
 - considerations 102,103
 - slack bytes 133-135
 - SYNCHRONIZED clause 133-135
 - VALUE clause 142
- depth of a report page 258-260
- descending sequence
 - in sort 243,244
 - in table handling 295,296
- descending sequence in merge (DOS/VS) xxv,xxvi
- DETAIL report group
 - description 261-263
 - GENERATE statement 273,274
 - LINE clause 263-265
 - NEXT GROUP clause 265,266
 - SUM counters 272,273
 - TYPE clause 267-269
- detail reporting 273,274
- device type specification 70
- DFR OCR macro instruction (DOS-VS) xxvii
- difference (in subtraction) 185,186
- direct access device (see mass storage device)
- direct data organization 59,60
- direct files
 - accessing techniques 60,61
 - ACTUAL KEY clause 79-81
 - APPLY EXTENDED-SEARCH clause 87
 - ASSIGN clause 69-72
 - BLOCK CONTAINS clause 100,101
 - error processing 175-178
 - initiating access 206-208
 - invalid key condition
 - READ statement 211,212
 - REWRITE statement 218,219

WRITE statement 213,217,218
 random access 78,61
 READ statement 211,212
 recording mode 103,104
 REWRITE statement 218,219
 sequential access 78,60
 user labels
 description 105,106
 processing 170,171
 WRITE statement
 description 217,218
 format 213
 direct indexing 298,290
 DISPLAY statement 220,221
 DISPLAY usage
 alignment 131
 alphabetic items 137
 alphanumeric items 137
 ASCII considerations 357
 description 136,137
 edited items 137
 numeric items 137
 SYNCHRONIZED clause 131
 DISPLAY-ST usage 325-329
 DIVIDE statement 183
 division, arithmetic operation 183
 division by zero 183,180
 division header 51
 division operator 153,154
 division of a program, definition 46
 DLINT OCR macro instruction
 (DOS/VS) xxviii
 dollar sign (see currency symbol, dollar sign)
 DOS/VS COBOL features (DOS/VS)
 FIPS flagger xxviii-xxxiii
 merge facility xxi-xxvii
 miscellaneous
 considerations xxxiv-xxxvi
 OBJECT-COMPUTER paragraph ii-iv
 VSAM file processing iv-xxi
 WHEN-COMPILED special register i
 3886 OCR processing xxvii,xxviii
 DOS/VS miscellaneous considerations (DOS/VS)
 ACTUAL KEY clause for 3340 xxxv
 APPLY CYL-OVERFLOW clause for 3340 xxxv
 APPLY WRITE-VERIFY invalid for 3540 xxxv
 ASSIGN clause for 3340, 3540 xxxiv
 BLOCK CONTAINS clause for 3540 xxxv
 CLOSE statement for 3540 xxxvi
 ERROR declaratives GIVING option for 3540 xxxv
 file processing summary xxxiv
 OPEN statement for 3540 xxxv
 sort input/output files xxxvi
 WRITE statement for 3540 xxxvi
 double spacing
 printer page 214,215
 source program listing 323
 doubleword
 floating-point items 138
 SYNCHRONIZED clause 131
 USING operands 229
 dummy files 69
 duplication of names 48,49
 E, in
 external floating-point items 118,121-123
 floating-point numeric literals 41
 EBCDIC collating sequence (Extended Binary Coded Decimal Interchange Code) 159,244
 editing
 insertion
 fixed 126
 floating 127
 simple 125
 special 125
 replacement 128
 sign control symbols
 description 119-121,126
 in fixed insertion editing 126
 in floating insertion editing 127
 in sterling report items 327-329
 symbols
 in alphanumeric edited items 124
 in arithmetic statements 179
 description 119-121
 in numeric edited items 124
 in SUM counter description 272
 zero suppression 128
 editing character
 description 119-121
 insertion
 fixed 126
 floating 127
 simple 125
 special 125
 zero suppression and replacement 128
 EJECT statement 323
 elementary item
 (see also data description clauses)
 description 95,96
 renaming 144-146
 slack bytes 132-135
 SYNCHRONIZED clause 130,131
 ellipsis (...) in formats 54
 ELSE option
 IF statement 166-168
 nested IF statements 167,168
 ON statement 320,321
 END DECLARATIVES. 169,150
 end of file
 when reading 212
 when sorting 248
 end of page condition 213,215
 end of volume positioning 222-224
 ENTER statement 233
 entry point 226-228
 ENTRY statement 227,228
 Environment Division
 ASCII considerations 355,356
 Configuration Section
 OBJECT-COMPUTER paragraph 65
 SOURCE-COMPUTER paragraph 64
 SPECIAL-NAMES paragraph 65-67
 Input-Output Section
 FILE-CONTROL paragraph 68-83
 I-O CONTROL paragraph 84-89
 international considerations 330
 organization 63
 segmentation considerations 311,65
 sort considerations 238-241

- System/370 card devices 379-381
- System/370 instructions 65
- Environment Division, VSAM (DOS/VS) v-ix
- equal sign (=)
 - in COMPUTE statement 182
 - in relation condition 158
- equal size operands in a relation condition 160
- equivalents
 - reserved words and abbreviations 40
 - THROUGH and THRU 40
- error bytes 176,178
- error conditions, arithmetic operations (see arithmetic statements, SIZE ERROR option)
- ERROR procedure, VSAM (DOS/VS) (see EXCEPTION/ERROR procedure, VSAM)
- error processing declaratives
 - and ACTUAL KEY 79
 - description and format 175-178
 - and Linkage Section 177
 - with sort 245,246
 - table of capabilities 178
- error processing declaratives (DOS/VS)
 - GIVING option for 3540 xxxv
 - VSAM file processing xii,xiii
- evaluation rules
 - arithmetic expressions 153
 - compound conditions 163-165
 - IF statements 166-168
- EXAMINE statement
 - description and formats 201,202
 - with sterling items 330
- EXCEPTION/ERROR procedure, VSAM (DOS/VS)
 - CLOSE statement xxi
 - description and format xii,xiii
 - OPEN statement xv
 - READ statement xvii
 - WRITE statement xviii
- execution, order of 150
- EXHIBIT statement 318-320
- exit point for procedures
 - error processing 176
 - EXIT statement 196,197
 - label handling 171
 - PERFORM statement 190
 - sort input/output procedures 245-249
- EXIT PROGRAM statement
 - description and format 232,231
 - symbolic debugging 365
- EXIT statement
 - description and format 196,197
 - PERFORM statement 190
 - PROGRAM option 232,231
 - with sort procedures 248,249
- explanatory comments
 - in every division 52,234
 - in Procedure Division 233,234
- exponent
 - floating-point items 137,138
 - floating-point numeric literals 41
- exponentiation operation 153,154
- Extended Binary Coded Decimal Interchange Code (EBCDIC)
 - collating sequence 159,244
 - nonnumeric literals 41
- extended search
 - for direct files 87
 - when reading 212
- EXTENDED-SEARCH option of the APPLY clause 87
- extended source program library facility 316,317
- external data 93
- external decimal items
 - class test 156
 - collating sequence for sort 244
 - description 137
 - internal representation 140
 - MOVE statement 200
 - relation condition 161
 - USAGE clause
 - description 137,140
 - format 136
- external floating-point items
 - collating sequence 244
 - description 137
 - internal representation 141
 - MOVE statement 200
 - relation condition 161
 - USAGE clause
 - description and format 137,136
 - internal representation 141
 - VALUE clause 143
- external-name in ASSIGN clause 71
- F-mode records
 - description 103,104
 - and OPEN REVERSED 207
 - recording mode 103,104
 - report print line 255
 - in sort 241
 - specification 104
- FD (see file description entry)
- figurative constants
 - description 43
 - EXAMINE statement 201,202
 - MOVE statement 200
 - relation condition 161
 - TRANSFORM statement 203-205
 - VALUE clause 142
- file
 - definition 93
 - disposition of
 - CLOSE statement 221-225
 - OPEN statement 206-208
 - FILE-CONTROL paragraph 68-83
 - file description entry 100-107
 - inter-record slack bytes 135,136
 - I-O-CONTROL paragraph 84-89
 - FILE-CONTROL paragraph
 - ACCESS MODE clause 78
 - ACTUAL KEY clause 79-81
 - ASSIGN clause
 - description and format 69-75
 - for sort 238,239
 - copying 313-315
 - FILE-LIMIT clause 77,78
 - format 68
 - NOMINAL KEY clause 82
 - PROCESSING MODE clause 78,79
 - RECORD KEY clause 83

RESERVE clause 77,238
SELECT clause
 description and format 69
 for sort 238,239
 sort considerations 238,239
TRACK-AREA clause 83
FILE-CONTROL paragraph (DOS/VS)
 merge facility xxii
 VSAM files v-viii
 3340 and 3540 devices xxiv,xxv
file description entry
 BLOCK CONTAINS clause 100,101
 description and format 100,97
 DATA RECORDS clause
 description and format 106,107
 report writer 255
 LABEL RECORDS clause 105,106
 RECORD CONTAINS clause
 description and format 102,103
 report writer 255
 RECORDING MODE clause 104,255
 REPORT clause 254,255
 report writer 254,255
 sort 241,242
 VALUE OF clause 106
file description entry, VSAM (DOS/VS) ix
file information area for OCR
 (DOS/VS) xxviii
FILE-LIMIT clause 77,78
file-name, definition 41
file processing technique
 general description 59-62
 input/output errors 175-178
 sample programs 33,34,336-340
 statements and clauses 352-354
 summary 59-62,352-354
file processing technique (DOS/VS)
 System/370 device summary xxxiv
 VSAM summary xiv
File Section
 ASCII considerations 356,357
 boundary alignment 131
 content 97,100-107
 copying 313-315
 file description entry 97,100-107
 format 97,100
 naming data 111
 record description entry 108,109,97
 report writer considerations 254,255
 sort considerations 241,242
file size for sort 249
FILE STATUS clause, VSAM (DOS/VS) v,viii
files sharing same storage area 85,86
FILLER
 CORRESPONDING option 179
 inter-record slack bytes 135,136
 record description entry 111,108
FINAL control
 CONTROL clause 257,258
 TYPE clause 267-269
final phase of sort 248
FIPS flagger description
 (DOS/VS) xxviii-xxxiii
fixed insertion editing 126
fixed length record format (see F mode records)
fixed point numeric item 122
fixed point numeric literal 42
fixed portion of a segmented program 309,311
floating insertion editing 127
floating-point data items (see external and internal floating-point items)
floating-point numeric literal
 description 42
 MOVE statement 200
flowchart
 nested IF statement 168
 PERFORM statements
 varying one identifier 193
 varying three identifiers 195
 varying two identifiers 194
 SEARCH statement 304
footing report groups
 CONTROL clause 257,258
 GENERATE statement 274
 NEXT GROUP clause 266
 PAGE LIMIT clause 259,260
 report group description 261
 SUM clause 272,273
 TYPE clause 267-269
form overflow (see end of page condition)
format
 DISPLAY statement output 220,221
 EXHIBIT statement output 318-320
 logical records 95,96
 reference 50-52
 report page 252-254
format control of the source program listing 323
format F records (see F-mode records)
format notation 53,54
format S records (see S-mode records)
format summary 341-350
format U records (see U-mode records)
format V records (see V-mode records)
fractions, internal floating-point items 138,141
FROM identifier option, VSAM (DOS/VS)
 description x,xi
 REWRITE statement xi
 WRITE statement xvii,xi
full FIPS flagging (DOS/VS) xxix-xxxii
fullword
 binary item 138
 slack bytes 132
 SYNCHRONIZED clause 131
function-name
 CODE clause
 description and format 256,257
 in SPECIAL-NAMES 66,67
 description 66,67,41
 in SPECIAL-NAMES paragraph 66,67
 in switch-status condition 162,66,67
 System/370 card devices 379,380
 in WRITE statement 214-217,66
GENERATE statement 273,274,253
generic key
 specification 208,209
GIVING option
 arithmetic statements
 ADD 181
 description 179

DIVIDE 183
 MULTIPLY 184
 SUBTRACT 185
 error handling declarative 175-178
 SELECT sentence format 238
 SORT statement
 description 247,243
 GIVING option (DOS/VS)
 merge statement xxvi
 3540 error processing xxxv,xxxvi
 glossary 385-397
 GO TO MORE-LABELS 171,172
 GO TO statement
 ALTER statement 188
 CALL statement 227
 debug packet 322
 description and formats 187,188
 error processing procedures 177
 IF statement 166
 label handling procedures 171,172
 PERFORM statement 190
 segmentation 312
 sort procedure 245,246
 GOBACK statement
 description and format 232,231
 symbolic debugging 365
 group
 collating sequence for sort 244
 contents 95,96
 report 261-263
 GROUP INDICATE clause 270,273
 group item
 description 95,96
 MOVE statement 200,199
 OCCURS clause 294
 relation condition 161
 renaming 144-146,112-115
 report 261-263
 slack bytes 132-136
 USAGE clause 136
 VALUE clause 143,144

 halfword
 binary item 138,140
 slack bytes 132
 SYNCHRONIZED clause 131
 halting execution
 STOP RUN 196
 subprograms linkage 231,232
 header labels
 description 105
 nonstandard 105,170
 standard 105,170
 user 105,170
 heading report groups
 CONTROL clause 257,258
 GENERATE statement 274
 NEXT GROUP clause 266
 PAGE LIMIT clause 259,260
 report group description 261
 SUM clause 272,273
 TYPE clause 267-269
 hierarchy
 arithmetic expressions 153
 controls in report writer 268,257
 qualification 48,49
 relations 163
 structure of a record 95,96
 high-intermediate FIPS flagging
 (DOS/VS) xxxi,xxxii
 HIGH-VALUE (HIGH-VALUES) figurative
 constant
 description 43
 MOVE statement 200
 permissible comparisons 161
 RECORD KEY 83
 TRANSFORM statement 203-205
 hyphen
 (see also minus symbol)
 collating sequence 159,244
 continuing lines 52
 program-names 58
 words 37

 I-O files
 CLOSE options 222-225
 error handling 175-178
 label handling 170-171
 OPEN statement 206-208
 REWRITE statement 218,219
 WRITE statement 213,217,218
 I-O-CONTROL entry (DOS/VS)
 merge facility xxii,xxiii
 VSAM files viii,ix
 3340 and 3540 devices xxxv
 I-O-CONTROL paragraph
 APPLY clause 87-89
 copying 313-315
 description and formats 84-89
 MULTIPLE FILE TAPE clause 86,87
 purpose 84
 RERUN clause 84,85
 SAME clause 85,86
 sort considerations 239-241
 IBM sterling representation 324
 ID Division header 57,46
 Identification Division
 comment-entry in 57,58
 DATE-COMPILED paragraph 58
 header 57,46
 PROGRAM-ID paragraph 57,58
 structure of 57
 identifier, definition 48
 identifying records
 data-name 111
 indexing 290,291
 qualification 48,49
 reports 261
 subscripting 289,290
 IF statement
 description and format 166-168
 nested 167,168
 IGN parameter of ASSGN job control
 statement 69
 imperative statements
 arithmetic 179-186
 data-manipulation 198-205
 declarative 169-178,276,277
 definition 150
 input/output 206-225
 procedure branching 187-197
 report writer 273-276

- sort 242-249
- table handling 301-306
- implied subjects and
 - relational-operators 164,165
- IN qualifier connective
- indexes 291
 - names 48
 - subscripts 290
- incrementing
 - index-name values 306,299
 - LINE-COUNTER special register 278
 - PAGE-COUNTER special register 277
 - SUM counters 272,273
- indentation of level numbers 97
- independent segment 309-312
- index data item
 - MOVE statement 198
 - relation condition 300,301,161
 - USAGE clause description 299,300
- index-name
 - description 297,41
 - modifying values in 306,299
 - MOVE statement 198
 - OCCURS clause 293,296-299
 - PERFORM statement 189,191,192
 - relation condition 300,301,161
 - SEARCH statement 301-305
 - SET statement 306
 - subprogram linkage and 236,229
 - value in 297-299
- INDEX option of the USAGE clause (see index data item)
- INDEXED BY option of the OCCURS clause (see index-name)
- indexed data organization 60
- indexed files
 - access techniques 61,62
 - APPLY clause 88,89
 - ASSIGN clause 69-71
 - blocking factor 101
 - error processing 175-178
 - index in core 89
 - initiating processing 206-209
 - invalid key condition
 - READ 211,212
 - REWRITE 218,219
 - START 208,209
 - WRITE 213,217,218
 - LABEL RECORD clause 105
 - NOMINAL KEY clause 82
 - OPEN statement 206-208
 - overflow areas 88
 - READ statement 211,212
 - RECORD KEY clause 83
 - recording mode 104
 - restrictions on processing of 208,61
 - REWRITE statement 218,219
 - START statement 208,209
 - VSAM (see VSAM file processing)
 - WRITE statement 213,217,218
- indexes used with qualifiers 290,291
- indexing tables
 - description 290,291
 - direct 298,290
 - INDEXED BY option 296-299
 - relative 299,291
- initializing
 - index values 306,191
 - report writer special registers 275
 - sort special registers 249
 - values of data items 142
- INITIATE statement 275,253
- initiating
 - access to a mass storage file 208-210
 - file processing 206-208
 - report processing 275
- input files
 - APPLY EXTENDED-SEARCH 87
 - effect of close options 222-225
 - error handling 175-178
 - inter-record slack bytes 135,136
 - intra-record slack bytes 132-135
 - label handling 170-172
 - OPEN statement 206-208
 - READ statement 211,212
 - record size 102
- input format for source programs 50-52
- input/output areas (buffers) 77
- input/output error (see error processing declaratives, invalid key condition)
- Input-Output Section
 - copying 313-315
 - description and format 68
 - File-Control paragraph 68-83
 - I-O-Control paragraph 84-89
 - sort considerations 238-241
- Input-Output Section (DOS/VS)
 - merge facility xxii,xxiii
 - sort feature xxxvi
 - VSAM files v-ix
 - 3340,3540 considerations xxxiv,xxxv
- input/output statements
 - ACCEPT 219,220
 - CLOSE 221-225
 - DISPLAY 220,221
 - OPEN 206-208
 - READ 211,212
 - REWRITE 218,219
 - SEEK 210
 - START 208,209
 - WRITE 212-218
- input/output statements, VSAM (DOS/VS)
 - CLOSE xxi
 - DELETE xx,xxi
 - OPEN xiii-xv
 - READ xvi,xvii
 - REWRITE xix,xx
 - START xv
 - WRITE xvii-xix
- input phase of merge (DOS/VS) xxv,xxvi
- input phase of sort 245-248
- input phase of sort (DOS/VS) xxxvi
- INSERT card for copying 316,317
- insertion
 - asterisks 128,120
 - commas 125,119
 - currency symbol 126,120
 - periods 125,119
 - sign control symbols 126,120
 - spaces
 - single insertion 125
 - symbols 118,119
 - zero suppression 128
 - zeros 125,119
- insertion character, and item size 118-120

- insertion editing
 - fixed 126
 - floating 127
 - simple 125
 - special 125
- INSTALLATION paragraph 57
- integer, description 42
- integer literals (see fixed-point numeric literals)
- inter-record slack bytes 135,136
- intermediate results
 - arithmetic statements 333,334
 - compound conditions 163,164
- internal data 98
- internal decimal items
 - allowable characters 139
 - class test 156
 - collating sequence 243,244
 - description 139
 - internal representation 141
 - MOVE statement 200
 - relation condition 161
 - SYNCHRONIZED clause 131
 - USAGE clause
 - description and format 139,136
 - internal representation 141
- internal floating-point items
 - collating sequence 244
 - description 138
 - internal representation 141
 - MOVE statement 200
 - relation condition 161
 - USAGE clause
 - description and format 138,136
 - internal representation 141
- internal representation
 - alphabetic and alphanumeric items 137
 - external decimal items 137,140
 - numeric items 140,141
 - sterling items 324
- international currency
 - considerations 330,65-67
- INTO/FROM identifier option, VSAM (DOS/VS)
 - description xi
 - READ statement xi,xvi
 - REWRITE statement xi
 - WRITE statement xi,xviii
- intra-record slack bytes 132-135
- INVALID KEY condition
 - READ 211,212
 - REWRITE 218,219
 - SEEK 210
 - START 208,209
 - WRITE 213,217,218
- INVALID KEY condition, VSAM (DOS/VS)
 - DELETE statement xx,xi
 - description xi
 - READ statement xvii,xi
 - REWRITE statement xx,xi
 - START statement xv,xi
 - WRITE statement xviii,xix,x
- INVALID KEY option (see invalid key condition)
- justification
 - and JUSTIFIED clause 116
 - and MOVE statement 199
- JUSTIFIED clause 116
- KEY clauses
 - ACTUAL 79-81
 - NOMINAL 82
 - RECORD 83
 - RECORD, VSAM (DOS/VS) vii,viii
- KEY option
 - MERGE statement (DOS/VS) xxv,xxvi
 - OCCURS clause 295,296
 - SEARCH ALL statement 302,305
 - SORT statement 243,244
 - START statement 208,209
- key words
 - description 40
 - format notation 53
- keys
 - for MERGE statement (DOS/VS) xxv,xxvi
 - for SORT statement 243,244
 - for START statement 208,209
 - for START statement (DOS/VS) xv
 - for table SEARCH
 - KEY option 295,296
 - SEARCH ALL statement 305,302
- label handling
 - ASCII considerations 356
 - CLOSE options 222-225
 - declarative 170-175
 - LABEL RECORDS clause 105,106
 - multivolume files 223
 - OPEN statement 206-208
 - sample program 172-175
 - for sort 241,242
 - USE sentence 170,171
 - writing 217
- LABEL RECORDS clause
 - ASCII considerations 356
 - description and format 105,106
 - label procedures 170,171
- LABEL RECORDS clause (DOS/VS)
 - VSAM ix
 - 3540 xxxv
- leading zeros, suppression 127,128
- left justification 116
- length
 - binary items 138
 - BLOCK CONTAINS clause 100,101
 - data-name in APPLY CORE-INDEX 89
 - external decimal items 137
 - external floating-point items 122,123
 - internal decimal items 139
 - internal floating-point items 138
 - RECORD CONTAINS clause 102,103
 - and standard data format 102
- level indicator
 - file description entry 97,100
 - reference format 51
 - report writer feature 254,256
 - sort feature 241
 - summary of 95
- level number
 - data description entry 111,108-110
 - indentation of 97
 - reference format 51
 - special 108-110
 - summary of 96,97
 - use 95-97

- level number 01 items
 - boundary alignment 131,229
 - CALL statement 229
 - copying 313-315
 - description 96
 - File Section
 - data description 108,109
 - REDEFINES restriction 112
 - word boundary 131
 - Linkage Section
 - data description 108,109
 - description 98,99
 - subprogram linkage 229
 - word boundary 131,132
 - reference format 51
 - Report Section 261-263,265-269
 - SYNCHRONIZED clause 130,131
 - Working-Storage Section 108,109,131
- level number 02-49 items
 - data description clauses 108,109
 - description 95,96
 - indentation 97
 - reference format 51
 - Report Section 261-263
 - slack bytes 132-135
 - SYNCHRONIZED clause 130
 - Working-Storage Section 108,109,98
- level number 66 items
 - description 96,109
 - format 108
 - indentation 97
 - RENAMES clause 144-146
 - restrictions 109
 - rules for use 109
- level number 77 items
 - called subprograms 229
 - copying 313-315
 - description 96
 - Linkage Section
 - description 99
 - subprogram linkage 229
 - word boundary 131
 - reference format 51
 - use 96,108,109
 - VALUE clause 142,143
 - Working-Storage Section 98,108,109
- level number 88 items
 - condition-name condition 157
 - description and format 109,110,97
 - indentation 97
 - rules for use 109,110
 - VALUE clause 142-144
- library facility (see source program library facility)
- library-name 313-315,41
- LINE clause 263-265
- line-control cards 364,365
- LINE-COUNTER special register
 - description 278
 - GENERATE statement 274
 - INITIATE statement 275
- lines in a report 258-260,263-266
- lines, spacing of
 - program output 213-217
 - report 263-266
 - source program 323,52
- Linkage Section
 - boundary alignment 130,131,229
 - content 98,99
 - copying 313-315
 - data item description entry 108,109
 - error declarative 177
 - format 99
 - intra-record slack bytes 132-135
 - record description entry 108,109
 - structure 94
 - use of FILLER 111
 - USING option of the CALL statement 229
 - VALUE clause 142
- Linkage Section and OCR processing (DOS/VS) xxviii
- linkage statements for subprograms (see subprogram linkage statements)
- list of compiler features
 - DOS/VS COBOL 11
 - Version 3 COBOL 11-13
- lister facility (DOS/VS) 11
- literal
 - CODE clause 256,257
 - description 41,42
 - as a function-name 66,67
 - MOVE statement 200
 - nonnumeric 42
 - numeric 42
 - permissible comparisons 161
 - VALUE clause 142-144
- location of slack bytes 132-136
- logical connectives in compound conditions 162-165,40
- logical operators 162,40
- logical record
 - description 93
 - redefining
 - description 112-115
 - File Section restriction 112
 - renaming 144-146
 - size of 102
 - slack bytes in 132-136
- long-precision internal floating point items (see internal floating-point items)
- low FIPS flagging (DOS/VS) xxxii,xxxiii
- low-intermediate FIPS flagging (DOS/VS) xxxii
- LOW-VALUE (LOW-VALUES) figurative constant
 - in ACTUAL KEY clause 80
 - description 43
 - MOVE statement 200
 - track identifier 80
 - TRANSFORM 203-205
- lowercase letters in ACCEPT (DOS/VS) xxxvi
- lower-case words in formats 53
- magnetic tape (see tape)
- magnitude of floating-point items 137
- main program, description 226,227
- major control break 257,258
- mantissa
 - description 137,138
 - internal representation 141
 - PICTURE clause 123
- mass storage devices
 - DOS/VS information (DOS/VS) iv,xxxiv-xxxvi

- error information 176,178
 - list of 70,72
- mass storage files
 - (see also VSAM file processing (DOS/VS))
 - CLOSE statement 221-225
 - OPEN statement 206-208
 - READ statement 211,212
 - REWRITE statement 218,219
 - SEEK statement 210
 - START statement 208,209
 - WRITE statement 213,217,218
- maximum length
 - arithmetic operands 181-184,186
 - binary items 138
 - data description entries 108
 - external decimal items 137,122
 - floating-point items 122,123,137
 - internal decimal items 138,122
 - items in EXHIBIT statement 318-320
 - keys in
 - sort 244
 - table handling 295
 - numeric edited items 124
 - PICTURE character string 117
 - record
 - CONSOLE 220
 - data 108
 - SYSLST 220
 - SYSPCH/SYSPUNCH 220
 - tables 293
- maximum length, keys in merge (DOS/VS) xxv
- maximum number
 - index-names 296
 - keys
 - sort 244
 - table handling 295
 - procedure-names in GO TO statement 187
- maximum number, keys in merge (DOS/VS) xxv
- maximum size (see maximum length)
- maximum value
 - ACTUAL KEY track identifier 80
 - binary item and PICTURE clause 118,138
 - floating-point items 137,42
 - NSTD-REELS special register 45
 - ON statement integer 321
 - RERUN clause integer 85
 - RESERVE clause integer 77
 - subscript 289
- MEMORY SIZE clause 65
- merge feature (DOS/VS)
 - FILE-CONTROL entry
 - ASSIGN clause xxii
 - SELECT clause xxii
 - I-O-CONTROL entry xxii,xxiii
 - input/output files valid xxii
 - MERGE statement
 - ASCENDING/DESCENDING KEY
 - option xxv,xxvi
 - file-names in xxv,xxvi
 - format xxv
 - GIVING option xxvi
 - OUTPUT PROCEDURE option xxvi,xxvii
 - segmentation restrictions xxvii
 - USING option xxvi
 - SAME clause xxii,xxiii
 - SD entry xxiv
- methods of data reference 48,49
- MFCU (multifunction card unit) support
 - (DOS/VS) xxxiv,xxxvi,11
 - minimum value
 - floating-point items 137,42
 - subscript 289
 - minor control break 268,257,258
 - minus symbol
 - (see also hyphen)
 - arithmetic expressions 153,154,39
 - collating sequence 159,244
 - external floating-point literals 42
 - indexing 299,290,291
 - PICTURE clause
 - description 120,121
 - external floating-point items 137,123
 - numeric edited items 124-128
 - sterling items 325-329
 - SIGN clause 129,130
 - as unary operator 153,154
 - miscellaneous considerations
 - (DOS/VS) xxxiv-xxxvi
 - mnemonic-name
 - ACCEPT statement 219,220
 - assignment of 66,67
 - CODE clause 256,257
 - DISPLAY statement 220
 - SPECIAL-NAMES paragraph 66,67
 - WRITE statement 213-216
 - mode F records (see F-mode records)
 - mode U records (see U-mode records)
 - mode V records (see V-mode records)
 - modification
 - library text
 - DELETE and INSERT cards 316,317
 - sort records
 - after sorting 246,247
 - before sorting 245
 - MOVE statement
 - CORRESPONDING option 198,199
 - description and formats 198-200
 - permissible moves 200
 - sort special registers 249
 - sterling items 330
 - multifunction card unit (MFCU) support
 - (DOS/VS) xxxiv,xxxvi,11
 - MULTIPLE FILE TAPE clause 86,87
 - multiple results
 - ADD statement 181
 - SUBTRACT statement 185
 - multiplication 184,153,154
 - multiplication operator 153,154,39
 - MULTIPLY statement 184
 - multivolume processing
 - ASSIGN clause 69,70
 - CLOSE statement 221-225
 - NSTD-REELS special register 69,70,45
 - reading 212
 - writing 217
 - name
 - data item 111
 - description of 41
 - field in system-name 71
 - indexing of 290,291,296-299
 - procedure 41
 - qualification 48,49
 - record 111,41
 - subscripting of 289,290

NEGATIVE operand in a sign condition 162
 negative value
 DISPLAY statement 221
 external floating point items 123
 numeric edited items 124-128
 PERFORM statement 191
 SIGN clause 129,130
 sign condition 162
 nested
 IF statements 167,168
 OCCURS clauses 293-295
 PERFORM statements 190
 REDEFINES clauses 113
 NEXT GROUP clause
 description and format 265,266
 effect of PRINT-SWITCH 277
 NOMINAL KEY clause
 description and format 82
 indexed files 61,62
 READ statement 211,212
 REWRITE statement 218,219
 START statement 208,209,82
 WRITE statement 217,218,212
 noncontiguous data items
 (see level number 77 items)
 nonnumeric literals
 continuation of 52
 description 42
 EXAMINE statement 201,202
 MOVE statement 200
 relation condition 161
 TRANSFORM statement 203-205
 VALUE clause 142-144
 nonnumeric operands
 MOVE statement 199,200
 relation condition 159-161
 nonstandard labels
 declarative 170-175
 GO TO MORE-LABELS 171,172
 LABEL RECORDS clause 105,106
 multiple reel files 69,70,45
 NSTD-REELS special register 45,69,70
 reversed reading 86,87,207
 sample program 173-175
 system procedures 170,171
 NOT condition construction
 abbreviated conditions 164,165
 compound conditions 162-165
 test conditions 155
 NOT logical operator
 description 162-165
 implied subjects and relational
 operands 164,165
 NOTE statement 233,234
 NSTD-REELS special register 45,69,70
 null report group 261
 numeric character
 description 119-121
 PICTURE clause 119-128
 numeric class test 156
 numeric data item
 BLANK WHEN ZERO clause 115
 class test 156
 EXAMINE statement 201,202
 fixed-point
 binary 138-140
 external decimal 137,140
 internal decimal 139,141
 floating-point
 external
 description 137,122,123
 internal representation 141
 internal 138,141
 internal representation 140,141
 MOVE statement 199,200
 PICTURE clause 122,123
 relation condition 161,159
 VALUE clause 142-144
 numeric data item (DOS/VS)
 FILE STATUS clause, VSAM viii
 RECORD KEY clause, VSAM vii,vii
 numeric edited items
 BLANK WHEN ZERO clause 115
 description 124-128
 MOVE statement 199,200
 relation condition 159-161
 USAGE clause 137
 numeric item, description 122,123
 numeric literal
 continuation 52
 description 42
 MOVE statement 200
 relation condition 161
 VALUE clause 142-144
 numeric operands
 ADD statement 181,182
 COMPUTE statement 182
 DISPLAY statement 221
 DIVIDE statement 183
 EXAMINE statement 201,202
 MOVE statement 199,200
 MULTIPLY statement 184
 relation conditions 159,161
 SUBTRACT statement 185,186
 OBJECT-COMPUTER paragraph
 computer-name 65
 copying 313-315
 description and format 65
 SEGMENT-LIMIT clause 311,65
 System/370 instruction generation 65
 OBJECT-COMPUTER paragraph (DOS/VS) ii-iv
 object of a condition 158
 object program definition 57
 OCCURS clause
 description and formats 292-299
 direct indexing 298
 redefining restriction 112
 relative indexing 299
 renaming restriction 145
 slack bytes 133-135
 value restriction 142
 OCR processing (DOS/VS) xxvii,xxviii
 OF qualifier connective
 with indexes and subscripts 289-291
 with name 48,49
 OFF/ON STATUS clause 66,67
 OMR (optical mark read) processing 72,76
 ON statement 320,321
 Version 3 improvement 320,321
 ON/OFF STATUS clause 66,67
 OPEN statement
 declaratives 170-175
 description and format 206-208

OPEN statement, VSAM (DOS/VS)
 description and format xiii-xv
 PASSWORD clause xiv,viii
 processing capabilities xiv
 status key xiv,x
 OPEN statement for 3540 (DOS/VS) xxxvi
 operands, overlapping
 arithmetic statements 180
 MOVE statement 199
 TRANSFORM statement 203
 operation sign (see sign, SIGN clause)
 operator communication 196,219-221
 optical character reader
 (DOS/VS) xxvii,xxviii
 optical mark read processing 72,76
 optimized object code, Version 3 12
 optional words in formats 53
 OR logical operator in compound
 conditions 162-165
 order of evaluation for compound
 conditions 163,164
 order of execution 150
 organization
 COBOL program 46,47
 data 59-62,93
 Data Division 94
 Data Division entries 95-97
 Environment Division 63
 field of system-name 71-75
 Identification Division 57
 Procedure Division 149,150
 ORGANIZATION clause, VSAM (DOS/VS)
 description and format vii,v
 processing capabilities xiv
 output files
 CLOSE options 222-225
 error handling 175-178
 inter-record slack bytes 135,136
 intra-record slack bytes 132-135
 label handling 170-175
 OPEN statement 206-208
 sample programs 336-338
 WRITE statement 212-218
 output files, VSAM (DOS/VS)
 OPEN statement xiii-xv
 processing capabilities xiv
 WRITE statement xvii-xix
 output phase of merge (DOS/VS) xxv-xxvii
 output phase of sort 246-248
 output source listing format
 compiler 50-52
 control of 323
 overflow records 88
 overlapping data groupings 144-146
 overlapping operands
 arithmetic statements 180
 MOVE statement 199
 TRANSFORM statement 203
 overlayable fixed segment 309-311
 overlaying procedures 309-312

 padding in a physical record
 BLOCK CONTAINS clause 100,101
 slack bytes 132-136
 page change in a report
 GENERATE statement 274
 LINE clause 263-265
 NEXT GROUP clause 265,266
 PAGE LIMIT clause 258-260
 PAGE clause (see PAGE LIMIT clause)
 page condition 258-260
 PAGE-COUNTER special register
 description 277
 GENERATE statement 274
 INITIATE statement 275
 PAGE FOOTING report group
 description 268
 LINE clause 263-265
 NEXT GROUP clause 265,266
 PAGE LIMIT clause 258-260
 TERMINATE statement 275,276
 TYPE clause 267-269
 page format in Report Writer 258-260
 PAGE HEADING report group
 definition 267
 GENERATE statement 273,274
 LINE clause 263-265
 NEXT GROUP clause 265,266
 PAGE LIMIT clause 258-260
 TYPE clause 267-269
 PAGE LIMIT clause 258-260
 pairing
 ELSE in nested IF statements 167,168
 parentheses in arithmetics
 expressions 153,154
 parentheses in subscripts and
 indexes 289-291
 symbols in arithmetic expressions 154
 symbols in compound conditions 164
 paragraph
 DATE-COMPILED 58
 definition 149
 FILE-CONTROL 68-83
 I-O-CONTROL 84-89
 OBJECT-COMPUTER 65
 Procedure Division 149,51
 PROGRAM-ID 57,58
 SOURCE-COMPUTER 64
 SPECIAL-NAMES 65-67
 paragraph-name
 qualification 48,49
 reference format 51
 rules for forming 149,41
 parameters for OCR processing
 (DOS/VS) xxviii
 parentheses
 arithmetic expressions 153,154
 conditions 163,164,155
 PICTURE clause 118
 punctuation rules 38
 subscripting and indexing 289-291
 parity checking 88
 PASSWORD clause, VSAM (DOS/VS)
 description and formats viii,v
 OPEN statement xiv
 pence
 internal representations 324
 nonreport items 325,326

- report items 327-329
- PERFORM statement
 - CALL statement 227
 - debug packets 322
 - declarative section 171,176,277
 - description and formats 189-195
 - flowcharts 193-195
 - segmentation 312
 - sort procedures 245,246
- period
 - and comma exchanged 66,67,330
 - data description entry 108,109
 - division header 51
 - END DECLARATIVES 169,150
 - fixed-point numeric literals 42
 - floating-point numeric literals 42
 - paragraph-name 149,51
 - PICTURE clause
 - description 119,121
 - external floating-point items 123
 - indicated by P 119
 - indicated by V 119
 - numeric edited items 124-128
 - sterling items 325-329
 - section-header 149,51
 - section-name 149,51
 - sentence 149
- permanent segment 309-311
- permissible
 - comparisons 161
 - moves 200
 - symbol pairs
 - arithmetic expressions 154
 - compound conditions 164
- physical file, definition 93
- physical record
 - definition 93
 - size specification 100,101
- PICTURE clause
 - allowable symbols 118-120
 - ASCII considerations 357,361
 - categories of data
 - alphabetic 120
 - alphanumeric 120
 - alphanumeric edited 124
 - numeric 122,123
 - numeric edited 124-128
 - table of 117
 - character string 117,118
 - classes of data 117
 - editing
 - fixed insertion 126
 - floating insertion 127
 - simple insertion 125
 - special insertion 125
 - zero suppression and replacement 128
 - format 117
 - precedence of symbols in 121
 - repetition of symbols 118
- plus symbol
 - arithmetic expressions 153,154
 - collating sequence 159,244
 - indexing 290,291,299
 - PICTURE clause
 - description 120,121
 - external floating-point items 122,123
 - fixed insertion editing 126
 - floating insertion editing 127
 - numeric edited items 124-128
 - precedence in 121
 - sterling items 325-329
 - SIGN clause 129,130
 - unary operator 153,154
 - pocket select characters
 - combined function processing 379,383
 - definition 66
 - WRITE statement 214-216
 - positioning a file
 - CLOSE statement 222-224
 - MULTIPLE FILE TAPE clause 86,87
 - OPEN statement 206-208
 - POSITIVE operand in sign condition 162
 - positive value
 - external floating-point items 122,123
 - PERFORM statement 191
 - sign condition 162
 - unsigned operands 162,199
 - pound-report-string 327
 - pound-separator-string 327
 - pound sign
 - report item 327,329
 - representation 329
 - print line size for report 255
 - PRINT-SWITCH
 - description 277
 - INITIATE statement 275
 - NEXT GROUP report groups 266
 - printer support, Version 3 13
 - printer support, DOS/VS 11
 - priority numbers
 - ALTER statement 312
 - called programs 312
 - description 310,311
 - information for use 310
 - PERFORM statement 312
 - section header 310,311
 - segment limit 311
 - procedure branching statements
 - ALTER statement 188
 - EXIT statement
 - description and format 196,197
 - subprogram linkage 231,232
 - GO TO statement 187,188
 - PERFORM statement 189-195
 - STOP statement 196,231
 - procedure, definition 149
 - Procedure Division
 - content 149-152
 - copying 313-315
 - definition 46
 - organization 149,150
 - Report Writer considerations
 - declarative 276,277
 - GENERATE statement 273,274
 - INITIATE statement 275
 - overall 252-254
 - TERMINATE statement 275,276
 - Sort considerations
 - EXIT statement 248,249
 - RELEASE statement 247,248
 - RETURN statement 248
 - SORT statement 242-247
 - statement list 151,152

statements (see compiler-directing
statements, conditional statements,
imperative statements)
sterling considerations 330
structure 150
table handling considerations
relation conditions 300,301
SEARCH statement 301-305
SET statement 306
USING option on the division
header 228-231,149,150
Procedure Division (DOS/VS)
merge facility xxv-xxvii
VSAM file processing
common processing facilities ix-xi
CLOSE statement xxi
DELETE statement xx,xxi
EXCEPTION/ERROR declarative xii,xiii
file processing capabilities xiv
OPEN statement xiii-xv
READ statement xvi,xvii
REWRITE statement xix,xx
START statement xv
WRITE statement xvii-xix
procedure-name, definition 41
procedures, Declarative 169-178,276,277
processing capabilities summary, VSAM
(DOS/VS) xiv
PROCESSING MODE clause 78,79
program-control card 364
PROGRAM-ID paragraph 57,58
program-name
description 58
subprogram linkage 226
program termination 196,231,232
punctuation character
in formats 53
in a source program 38
quadruple spacing of source program
listing 323
qualification
condition-names 144,48,49
description 48,49
index-names 290,291
names 48,49
subscripts 289,290
qualifier connective, definition 40
qualifier, definition 48
quotation mark option
default 37
nonnumeric literals 42
program-name 58
QUOTE (QUOTES) figurative constant
description 43
in EXAMINE 201
in MOVE 200
in relation condition 161
in TRANSFORM 203
quotient 183
random access
ACCESS MODE clause 78
CLOSE statement 224,225
definition 60
direct files 61
error processing 175-178
indexed files 61,62
READ statement 211,212
REWRITE statement 218,219
SEEK statement 210
WRITE statement 213,217,218
random access, VSAM (DOS/VS) vii
random file processing
effect of CLOSE options 224,225
function of a read 211,212
function of a REWRITE 218,219
function of a SEEK 210
function of a WRITE 212,217,218
summary charts 353,354
range of a PERFORM statement 190-195
range of values
condition-name 143,144
priority numbers 310,311
sequence numbers on DELETE card 316,317
RCE (read column eliminate)
processing 72,73,76
RD (see report description entry)
READ statement
description and format 211,212
error processing 178
READ statement, VSAM (DOS/VS)
description and formats xvi,xvii
processing capabilities xiv
reading backwards 207
reading nonstandard labels
ASSIGN clause 70,71
MULTIPLE FILE TAPE clause 86
OPEN statement 207
READY/RESET TRACE statement 318
receiving data item
justification 116
MOVE statement 199,200
record
description 108,109
level number 95-97
naming 111
slack bytes
between records 135,136
within records 132-135
RECORD CONTAINS clause
description and format 102,103
Report Writer 255
Sort 241,242
RECORD CONTAINS clause, VSAM (DOS/VS) ix
record description entry
contents 108,109
definition 96
File Section 97
format 108
Linkage Section 99
maximum length 108
Report Writer 255
Sort records 241,242
Working-Storage Section 98
(see also data description clauses)
RECORD KEY clause 83
RECORD KEY clause, VSAM (DOS/VS)
description and format vii,viii,v
START statement xv
record length for sort records 241,242
record size default
ACCEPT statement 219
DISPLAY statement 220
report writer 255

- recording mode
 - ASCII considerations 357
 - default 103,104
 - specification 104
 - types 103,104
- RECORDING MODE clause
 - ASCII considerations 357
 - description and format 102
- RECORDING MODE clause invalid, VSAM (DOS/VS) ix
- REDEFINES clause
 - APPLY CORE-INDEX option 89
 - description and format 112-115
 - position when used 109,112
 - SYNCHRONIZED clause 131
 - VALUE clause 143
- reference format 50-52
- reformatted (lister) source listing (DOS/VS) 11
- registers (see special registers)
- regrouping data items
 - REDEFINES clause 112-115
 - RENAMES clause 144-146
- relation character
 - definition 39
 - in relation conditions 158
- relation conditions
 - ASCII considerations 358-360
 - characters used 158,39
 - condition-name as abbreviation 157
 - description and format 158-161
 - implied subject and relational operators 164,165
 - operands allowed 161
 - table handling 300,301
- relational-operators
 - compound conditions 162-165
 - definition 39
 - implied 164,165
 - relation condition 158
- relative indexing 290,291,299
- relative LINE clause 263-265
- RELEASE statement in sort 247,248
- relocation factor, Version 3 13
- remainder, definition 183
- REMARKS paragraph 57
- RENAMES clause
 - data description entry 108,109
 - description and format 144-146
 - level number summary 96
- renaming
 - data items (see REDEFINES clause, RENAMES clause)
 - logical records 112,97
- repetition of symbols in PICTURE 118
- replacement editing 128
- replacing zero with a space
 - BLANK WHEN ZERO clause 115
 - editing rules 124-128
 - PICTURE clause symbol 119
- REPORT clause 254,255,100
- REPORT clause invalid, VSAM (DOS/VS) ix
- report, description 252-254
- report description entry
 - CODE clause 256,257
 - CONTROL clause 257,258
 - COPY statement 313-315
 - definition 256
 - GENERATE statement 273,274
 - PAGE LIMIT clause 258-260
- report file, definition 254
- REPORT FOOTING report group
 - description 267-269
 - LINE clause 263-265
 - NEXT GROUP clause 265,266
 - PAGE LIMIT clause 258-260
 - TERMINATE statement 275,276
 - TYPE clause 267-269
- report group description entry
 - COLUMN clause 269
 - COPY statement 313-315
 - description and formats 261-263
 - GROUP INDICATE clause 270
 - LINE clause 263-265
 - NEXT GROUP clause 265,266
 - RESET clause 270,271
 - SOURCE clause 271
 - SUM clause 271-273
 - TYPE clause 267-269
 - USAGE clause 269
 - VALUE clause 273,271
- report groups
 - definition 261
 - page format 259
 - sequence of printing 268,269
 - types 267-269
 - USE sentence 276,277
- REPORT HEADING report group
 - description 267-269
 - GENERATE statement 274
 - LINE clause 264
 - NEXT GROUP clause 266
 - PAGE LIMIT clause 259
 - TYPE clause 267,268
- report page format effect on
 - LINE-COUNTER special register 278
 - PAGE-COUNTER special register 277
 - PAGE LIMIT clause 258-260
- Report Section
 - content 256-273
 - COPY statement 313-315
 - formats
 - report description entry 256
 - report group description entry 261,262
 - structure 94
 - VALUE clause
 - description and format 273,271
 - overall description 142
- Report Writer
 - Data Division considerations
 - File Section 254,255
 - overall description 252-254
 - Report Section 256-273
 - Procedure Division considerations
 - declarative 276,277
 - GENERATE statement 273,274
 - INITIATE statement 275
 - overall description 253,254
 - TERMINATE statement 275,276
 - USE statement 276,277
 - sample program
 - coding 279-283
 - output 284-288
 - special registers 277,278

- report-name
 - definition 41
 - qualifier for SUM counter 261
 - specification 256,273,274
- RERUN clause
 - ASCII considerations 356,361
 - processing programs 84,85
 - sort feature 240,361
- RERUN clause, VSAM (DOS/VS) viii,ix
- required words in formats 53
- RESERVE clause
 - description and format 77
 - System/370 card devices 381
- RESERVE clause, VSAM (DOS/VS) v,vi
- reserved words
 - definition 40
 - in formats 53
 - list of 344-346
- RESET clause, Report Writer 270,271
- RESET TRACE statement 318
- restarting a program 84,85,240
- retrieving an indexed file
 - access methods 61,62
 - READ statement 211,212
 - restriction 208
 - START statement 208,209
- return code
 - multivolume files 69,70
 - sort 249
- return code, VSAM status key (DOS/VS) x
- return from merge (DOS/VS)
 - GIVING option xxv,xxvi
 - output procedure xxv-xxvii
- return from sort
 - GIVING option 247
 - output procedure 246,247
- RETURN statement in merge (DOS/VS) xxvi
- RETURN statement in sort 248
- reversed reading of a file 207
- rewinding a tape file
 - CLOSE statement 222-224
 - OPEN statement 206,207
- REWRITE statement
 - description and format 218,219
 - error processing 178
- REWRITE statement, VSAM (DOS/VS)
 - description and format xix,xx
 - processing capabilities xiv
- rewriting mass storage files 218,219,60-62
- right justification 116
- rolling counters forward 272
- ROUNDED option in arithmetic statements
 - ADD 181
 - COMPUTE 182
 - description 179,180
 - DIVIDE 183
 - MULTIPLY 184
 - SUBTRACT 185
- rounding 179,180

- S, in PICTURE clause
 - class test 156
 - description 119,121
 - numeric items 122,123
 - precedence 121
 - SIGN clause 129-130
 - sterling nonreport items 326
- S-mode records
 - BLOCK CONTAINS clause 101
 - description 103,104
 - RECORDING MODE clause 104
- SAME clause
 - description and format 85,86
 - sort considerations 240,241
- SAME clause (DOS/VS)
 - merge facility xxii,xxiii
 - VSAM files viii,ix
- sample programs
 - creation of a direct file 336,337
 - creation of an indexed file 338
 - report writer 279-288
 - retrieval of a direct file 33,34
 - retrieval of an indexed file 339,340
 - sort 250,251
 - table handling 307,308
- scaling, effect on rounding 180
- scaling position character (P) 119,121
- scientific decimal item (see external floating-point items)
- SEARCH statement
 - description and formats 301-305
 - flowchart 304
- section
 - classification in segmentation 310,311
 - definition 149
 - format 150,310
- section header 149,51
- section-name 150,310
- SECURITY paragraph 57
- SEEK statement 210
- segment classification 310,311
- SEGMENT-LIMIT clause 311
- segmentation
 - ALTER statement 312
 - called programs 312
 - classifying segments 310,311
 - control of 310
 - fixed portion 309,311
 - GO TO statement 312
 - independent segments 309,311
 - overlayable fixed segments 309,311
 - PERFORM statement 312
 - permanent segments 309,311
 - priority numbers 310,311
 - program organization 309
 - restrictions on program flow 312
 - segment limit 311
- segmentation merge restrictions (DOS/VS) xxvii
- SELECT clause
 - COPY statement 313-315
 - description and format 69
 - GIVING option of SORT statement 238,239
 - sort work units 239
- SELECT clause (DOS/VS)
 - merge facility xxii
 - VSAM files v,vi
- semicolon
 - data description entry 109
 - separating statements 149
 - source program 38
 - SPECIAL-NAMES paragraph 66
- sentence
 - conditional 166

- description 149
- termination 149
- SEPARATE CHARACTER option of SIGN
 - clause 129,130
- separator
 - statements 149
 - sterling items 327-329
 - words 40
- sequence
 - entries in Data Division 100,109
 - entries in Environment Division 68,84
 - entries in formats 46
 - execution of Procedure Division 150
 - execution of segmented programs 310
 - report groups 261
 - sorting 243,244
 - table entries 295,296
- sequence checking compilation default 50
- sequence-number-field for copying 316,317
- sequence number in a source program 50
- sequential access
 - ACCESS MODE clause 78
 - ACTUAL KEY clause 79
 - APPLY WRITE-ONLY clause 87
 - ASSIGN clause 69-75
 - description 60
 - direct files 60
 - indexed files 61
 - NOMINAL KEY clause 82
 - recording mode 103-104
 - RECORDING MODE clause 104
 - RESERVE clause 77
 - sequential files 60
 - size of records 103,104
- sequential data organization 59
- sequential files (see standard sequential files)
- sequential multivolume files
 - effect of CLOSE options 222-224
 - reading 212
- sequential processing 60,61
- sequential single-volume files, effect of CLOSE options 222-224
- sequential VSAM files (see VSAM file processing)
- serial search of a table 302-304
- series connective, definition 40
- SET statement
 - description and formats 306
 - with index data items 299,300
 - with indexes 296-299,290
- setting values for index-names 306
- shading in text, explained (see Preface)
- sharing storage between files 85,86
- shilling representation
 - BSI and IBM conventions 324
 - internal 325,326
 - PICTURE clause 325,327
 - report items 327-329
- short-precision internal floating-point items (see internal floating-point items)
- sign
 - class condition 156
 - internal representation 140,141
 - literals 42
 - MOVE statement 199
 - PICTURE clause 119,121
 - relation condition 159
 - SIGN clause 129,130
 - sign condition 162
 - sterling items 326-329
 - subscripts 289
 - unary operator 153,154
- SIGN clause
 - ASCII considerations 357,361
 - class condition 156
 - description and format 129,130
 - Version 3 feature 13
- sign condition 162
- simple insertion editing 125
- single digit level number 96
- single spacing
 - printer page 214
 - source program listing 323
- SIZE ERROR option in arithmetic statements
 - ADD 181,182
 - COMPUTE 182
 - description 180
 - DIVIDE 183
 - MULTIPLY 184
 - SUBTRACT 185
- SKIP statements 323
- slack bytes
 - description 132-136
 - elementary computational items 132,133
 - group items containing an OCCURS
 - clause 133-135
 - inter-occurrence 133-135
 - inter-record 135,136
 - intra-occurrence 133
 - intra-record 132-135
 - physical record size 101
- small letters in ACCEPT (DOS/VS) xxxvi
- sort
 - ascending and descending
 - sequence 243,244
 - ASCII considerations 360,361
 - checkpoints 240
 - collating sequence 244
 - control of procedures 247
 - Data Division considerations 241,242
 - Environment Division considerations
 - File-Control paragraph 238,239
 - I-O-Control paragraph 239-241
 - file description entry 241,242
 - final phase 246,247
 - input phase 245,246
 - keys 243,244
 - modification of records 245-247
 - optimizing performance 249
 - Procedure Division considerations
 - EXIT statement 248,249
 - RELEASE statement 247,248
 - RETURN statement 248
 - SORT statement 242-247
 - sample program 250,251
 - special registers 249
 - work units 239
- sort enhancements (DOS/VS) xxxvi
- sort-file
 - COPY statement 313-315
 - description entry 241,242
 - SELECT clause 239
- sort special registers 249
- SORT statement
 - description and format 242-247

GIVING option 247
 INPUT PROCEDURE option 245
 OUTPUT PROCEDURE option 246,247
 RELEASE statement 247,248
 RETURN statement 248
 USING option 246
 SORT statement enhancement (DOS/VS) xxxvi
 SORT-OPTION clause (DOS/VS) xxxvi
 SORTWK1, SORTWK2 in sort 239
 SOURCE clause
 description and format 271
 report groups 269
 SOURCE-COMPUTER paragraph 64
 source program
 definition 57
 reference format 50-52
 structure 46,47
 source program library facility
 COPY statement 313-315
 extended
 BASIS 316
 DELETE and INSERT 316,317
 space
 alphabetic items 120
 BLANK WHEN ZERO clause 115
 collating sequence 159,244
 in editing 124-128
 external floating-point items 122,123
 replacement character 118,119,121
 simple insertion editing 125
 word separator 40
 space insertion 125-128,115
 SPACE (SPACES) figurative constant
 definition 43
 EXAMINE statement 201
 MOVE statement 200
 permissible comparisons 161
 TRANSFORM statement 203-205
 spacing program output 213-217
 spacing source program listing 323,52
 special characters
 CURRENCY-SIGN clause 67
 in formats 53
 special insertion editing 125
 special level numbers 96,97,108-110
 special-names, definition 41
 (see also mnemonic-name)
 SPECIAL-NAMES paragraph
 COPY statement 313-315
 description and format 65-67
 special register WHEN-COMPILED (DOS/VS) i
 special registers
 report writer
 LINE-COUNTER 278
 PAGE-COUNTER 277
 PRINT-SWITCH 277
 sort
 SORT-CORE-SIZE 249
 SORT-FILE-SIZE 249
 SORT-MODE-SIZE 249
 SORT-RETURN 249
 system
 COM-REG 44
 CURRENT-DATE 44
 NSTD-REELS 45,70
 TALLY (see TALLY special register)
 TIME-OF-DAY 44
 stacked items in formats 53
 standard data format
 alphabetic items 120
 alphanumeric items 120,124
 description 102
 logical records 102
 numeric items 122,124
 physical records 101
 standard labels 105,170,171
 STANDARD option of the LABEL RECORDS
 clause 105
 standard sequential file
 APPLY WRITE-ONLY clause 87
 ASSIGN clause 69-75
 BLOCK CONTAINS clause 100,101
 definition 59
 effect of CLOSE options 222-224
 error declarative capabilities 178
 labels 105,106
 reading 211,212
 recording mode 103,104
 RESERVE clause 77
 writing 212-217
 standard system procedures
 error routines 175-178
 label handling 170,171
 START statement
 description and format 208,209
 error processing 178
 NOMINAL KEY 82,61
 START statement, VSAM (DOS/VS)
 description and format xv
 processing capabilities xiv
 statement categories
 compiler-directing, list 152
 conditional, list 151
 description 150
 imperative, list 151,152
 statement number option, Version 3 13
 status key, VSAM (DOS/VS)
 CLOSE statement xxi
 DELETE statement xxi
 description viii,x
 EXCEPTION/ERROR declarative xii
 OPEN statement xiv
 READ statement xvi
 REWRITE statement xix
 START statement xv
 table of possible values x
 WRITE statement xviii
 sterling currency
 international considerations 330
 nonreport items
 description and format 325,326
 MOVE statement 200
 relation condition 161
 sign representation 326
 Procedure Division considerations 330
 report items
 description and format 327-329
 MOVE statement 200
 relation condition 161
 representations 324
 STOP RUN statement 196,231,232
 STOP statement
 calling and called programs 231,232
 format and description 196
 storage available for sort 249

- structure of
 - COBOL language 37-45
 - COBOL program 46,47
 - Data Division 94
 - Environment Division 63
 - Identification Division 57
 - Procedure Division 150
- subdivisions of data records
 - data description entry 108,109
 - description 95,96
- subject
 - of a condition
 - explicit 158,164,165
 - implied 164,165
 - of a data description entry 111
 - of an OCCURS clause 293
- subprogram, description 226
- subprogram linkage statements
 - CALL 226,227
 - ENTRY 227,228
 - EXIT PROGRAM 232
 - GOBACK 232
 - STOP RUN 196
 - termination considerations 231
 - USING option 228-231
- subroutine for OCR processing (DOS/VS) xxviii
- subscripts
 - condition-name 144
 - description and formats 289,290
 - qualification 290
 - qualifier 289,290
 - restrictions on use 291
- substitution
 - asterisk for zero 128,120
 - comma for period 66,67,330
 - dollar sign 66,67,330
 - space for zero
 - BLANK WHEN ZERO clause 115
 - in editing 127,128
 - PICTURE symbol 119,121
- subtotaling in a report 272,273
- SUBTRACT statement
 - CORRESPONDING option 179
 - description and formats 185,186
 - GIVING option 179
 - overlapping operands 180
 - ROUNDED option 179,180
 - SIZE ERROR option 180
- subtraction operator 153,154,39
- SUM clause 272,273
- SUM counter
 - definition 272
 - GENERATE statement 274
 - INITIATE statement 275
 - report-name only
 - qualifier for 261
 - RESET clause 270,271
 - resetting to zero
 - GENERATE statement 274
 - INITIATE statement 275
 - RESET clause 270,271
 - SUM clause 272,273
- summary reporting 273,274
- summation in a report 270-275
- suppress spacing 214,215
- suppression of
 - library entry listing 314
 - report group printing 277
 - sequence checking 50
 - spacing in WRITE statement 214,215
 - zeros in PICTURE clause 118-121,127,128
- suppression and replacement editing 128
- suppression symbols 118-121
- switch-status condition
 - description and format 162
 - function-names 162,66,67
- symbol pairs
 - arithmetic expressions 154
 - compound conditions 164
 - PICTURE clause 121
 - subscripting and indexing 289-291
- symbolic debugging, Version 3 13
- symbolic portion of ACTUAL KEY 80
- symbols
 - arithmetic expressions 153,154,39
 - floating-point literals 42
 - PICTURE clause 118-121
 - relation conditions 158,39
 - sterling currency formats 325
- SYNCHRONIZED clause
 - description and format 130,131
 - index data items 300
 - OCCURS clause 133-135
 - slack bytes 132-136
- SYSIPT 66,219,220
- SYSLST 66,220,221
- SYSPCH and SYSPUNCH 66,220,221
- system closing conventions 222-225
- system logical input device 66,219,220
- system logical output device 66,220,221
- system-name
 - ASSIGN clause 70-75
 - RERUN clause 84,85
 - sort feature 238-240
- system-name, VSAM (DOS/VS) vi,xxxiv
- system procedures (see standard system procedures)
- system special registers (see special registers)
- System/370 device support (DOS/VS) 11
- System/370 support, Version 3 65,11-13
- SYS001, SYS002 in Sort 239
- S01 through S05 function-names
 - combined function processing 379
 - SPECIAL-NAMES paragraph 66
 - WRITE statement 214,216

Table Handling

- ascending/descending sequence 295,296
- Data Division considerations
 - OCCURS clause 292-299
 - USAGE clause 299,300
- indexing 290,291,296-299
- Procedure Division considerations
 - relation conditions 300,301
 - SEARCH statement 301-305
 - SET statement 306
- sample program 307,308
- subscripting 289-291
- TALLY special register
 - ACCEPT statement 219
 - description 44
 - DISPLAY statement 221
 - EXAMINE statement 201,202
 - SOURCE clause 271

- subscripting 289
- SUM clause 271,273
- tape device, error information 176,178
- tape file, label handling
 - CLOSE statement 222-224
 - LABEL RECORDS clause 105,106
 - OPEN statement 207,208
 - READ statement 212
 - USE statement 170,171
- TERMINATE statement 275,276
- termination of
 - execution 196
 - main programs 231,196
 - report processing 275,276
 - sort processing 248,249
 - subprograms 231,232
- test conditions
 - class 156
 - compound 162-165
 - condition-name 157
 - description 155
 - relation 158-161
 - sign 162
 - switch-status 162
- THEN
 - used in IF statement 166
 - used in sentences 38,149
- THRU reserved word
 - and PERFORM statement 189,190
 - and THROUGH, equivalence of 40
 - and VALUE clause 142-144
- TIME-OF-DAY special register 44
- TRACE statement 318
- track address
 - algorithm example 336
 - component of ACTUAL KEY 80,81
 - direct file 59-61
- TRACK-AREA clause 83
- trailer labels 170,171,105
- transfer of control
 - ALTER statement 188
 - CALL statement 226,227
 - CALL statement for OCR (DOS/VS) xxviii
 - calling and called programs 226-232
 - DECLARATIVES 169
 - end of series of procedures 196,197
 - EXIT statement
 - description and format 196,197
 - sort procedures 248,249
 - subprogram linkage 231,232
 - GO TO statement 187,188
 - GO TO MORE LABELS 171,172
 - GOBACK statement 231,232
 - merge feature (DOS/VS) xxvi,xxvii
 - operating system 231,232,196
 - operator 220,196
 - PERFORM statement 189-195
 - RELEASE statement 247,248
 - RETURN statement 248
 - segments, among 310
 - sort feature 247-249
- TRANSFORM statement
 - ASCII considerations 358-360
 - description and format 203-205
- transmission errors 176
- triple spacing
 - printer page 213-216
 - source program listing 323
- truncation
 - alphabetic and alphanumeric items 199,116
 - arithmetic operations 180
 - of buffers 87
 - floating insertion editing 127
 - MOVE statement 199,116
 - numeric items 180,199
 - TYPE clause 267-269
- U-mode records
 - description 103,104
 - physical record size 100,101
 - specification 104
- UHL (user header label) 105
- unary + and - 153,154
- undefined record format (see U-mode records)
- unique names
 - indexing 290,291
 - qualification 48,49
 - subscripting 289,290
- unit, definition
 - in formats 54
 - in storage 222
- unit record volume
 - CLOSE options 222-224
 - description 222
 - error information 178
 - list 70,72
- unsigned numeric operands
 - considered positive 159,162
 - MOVE statement 199
 - relation condition 159
 - sign condition 162
- updating a file
 - REWRITE statement 218,219
 - sample programs 33,34,339,340
 - WRITE statement 213,217,218
- UPSI-0 through 7 (User Program Status Indicator bits) 67
- USAGE clause
 - alteration by redefining 114
 - ASCII considerations 357,361
 - default 136
 - description and formats 136-141
 - index data items 299,300
 - internal representations 140,141
 - sterling items 325-329
- use of coding form 50-52
- USE statement
 - declarative description and format 169
 - error processing 175-178
 - label processing 170-175
 - report writer 276,277
- user-created libraries 313-315
- user error procedures 175-178
- user header label (UHL) 105
- user labels
 - description 105
 - GO TO MORE-LABELS 171,172
 - procedures for handling 170-172
 - standard systems procedure
 - CLOSE statement 222-225
 - label processing 170,171

READ statement 212
 WRITE statement 217
 user program status indicator bits 67
 user trailer label (UTL) 105
 USING option in calling and called programs
 boundary alignment of identifiers 229
 in a CALL statement 226-229
 in a called program 227-229
 in a calling program 226-229
 in ENTRY statement 227-229
 index-names invalid 229,226
 on Procedure Division
 header 228,229,150
 USING option in merge (DOS/VS) xxvi
 USING option in sort 246
 USING option in sort (DOS/VS) xxxvi
 utility device
 class field in system-name 70
 list 70,72
 UTL (user trailer label) 105

V, in PICTURE clause
 description 119,121
 external floating-point items 123
 fixed point numeric items 122
 numeric edited item 124
 with P 119
 precedence 121
 sterling nonreport items 325,326
 V-mode records
 APPLY WRITE-ONLY clause 87
 description 103,104
 inter-record slack bytes 135,136
 intra-record slack bytes 132-135
 in sort 241
 specification 104
 specification of physical record
 size 100,101
 VALUE clause
 condition-names 142-144,109,110
 description and formats 142-144,109,110
 example 144
 report writer data items 271,273
 sterling items 326,329
 VALUE OF clause 106
 variable-length record format (see V-mode records)
 variable-length records
 recording mode 103,104
 size of print line in a report 255
 in sort 241
 variable length table 293-295,297,298
 verb profiles and statistics (DOS/VS) 11
 vertical positioning of a printed line 213-217
 volume positioning
 CLOSE statement 222-224
 OPEN statement 207
 volume switch
 CLOSE options 222-224
 label processing 170
 READ statement 212
 WRITE statement 217
 VSAM file processing (DOS/VS)
 Data Division considerations ix
 entry-sequenced data sets iv

FILE-CONTROL paragraph v-viii
 I-O-CONTROL paragraph viii-ix
 indexed files iv,v,xiv
 key-sequenced data sets iv
 overall description iv
 Procedure Division
 considerations ix-xxi
 processing capabilities summary iv,xiv
 sequential files iv,v,xiv
 description iv,v,xiv
 input to merge xxii
 input to sort xxxvi
 valid mass storage devices iv

WHEN-COMPILED special register (DOS/VS) i
 word
 characters used in 37
 continuation of 52
 definition 39,40
 separators 40
 types
 name 41
 reserved word 40
 special-name 41
 word boundary 138,131,132
 Working-Storage Section
 boundary alignment 131
 condition-name entries 142
 content 108-110,98
 COPY statement 313-315
 data item description entry 108-110
 formats 98,108,109
 level-numbers in 98,108
 naming data 111
 record description entry 98,108,109
 renaming entries in 144-146
 structure 94
 used in error processing 177
 use of FILLER 111
 values of items 142
 Version 3 support 13
 Working-Storage Section and OCR processing (DOS/VS) xxviii
 WRITE-ONLY option of the APPLY clause 87
 WRITE statement
 combined function processing 382-384
 description and formats 212-218
 error processing 176,178
 multivolume sequential files 217
 system-name organization field 218
 WRITE statement (DOS/VS)
 VSAM files
 description and format xvii-xix
 processing capabilities xiv
 3540 device xxxvi
 WRITE-VERIFY option of the APPLY clause
 description and format 88
 3540 device (DOS/VS) xxxv
 writing user labels 170-175,217

X, in PICTURE clause
 alphanumeric edited items 124
 alphanumeric items 120
 description 119,121
 precedence 121

Z, in PICTURE clause
 description 119,121
 numeric edited items 128,124
 precedence 121
 sterling report items 327-329
 zero suppression editing 128
 ZERO (ZEROES, ZEROS) figurative constant
 description 43
 EXAMINE statement 201
 MOVE statement 200
 relation condition 161
 replacing numeric literal 43
 TRANSFORM statement 203-205
 zero divisor 183,180
 zero insertion 119,121,124,125
 zero operand
 DIVIDE statement 183
 internal floating-point items 180
 relation condition 159,161
 sign condition 162
 zero, simple insertion editing 125
 zero suppression and replacement
 editing 128
 zone bits, external decimal items 137
 zoned decimal format 137,140

0, in PICTURE clause
 alphanumeric edited items 124
 description 119,121
 floating insertion editing 127
 numeric edited items 124
 precedence 121
 simple insertion editing 125

01-49 level numbers 96,109
 6, in sterling PICTURE clause 325
 66 level number
 definition 96
 description 144-146
 general description 108,109
 7, in sterling PICTURE clause 325,326
 77 level number 96,108,109
 8, in sterling PICTURE clause 325,326,
 88 level number
 definition 97
 description 143,144
 general description 109
 9, in PICTURE clause
 alphanumeric edited items 124
 description 119,121
 numeric edited items 124-128
 numeric items 122,123
 precedence 121
 sterling items 325-329
 2319, 3211, 3330, 3410, 3420, 3505, 3525
 support, Version 3 13
 2560, 3504, 3881 support, Version 3 12
 3203, 3340, 3540, 5203 (DOS/VS)
 processing support 11
 programming considerations xxxiv-xxx
 3886 OCR (DOS/VS)
 processing support 11
 programming considerations xxvii,xxv
 5425 MFCU (DOS/VS)
 processing support 11
 programming considerations xxxiv,xxx



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
921 United Nations Plaza, New York, New York 10017
(International)

IBM DOS Full American
National Standard COBOL
GC28-6394-6

Reader's
Comment
Form


Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

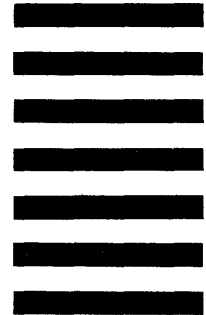

First Class Permit
Number 6090
San Jose, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
P. O. Box 50020
Programming Publishing
San Jose, California 95150



Fold and Staple



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM CORPORATION (File NO. S36U-24) Printed in U.S.A. GC28-6394-6



Technical Newsletter

~~J. D. PARKER~~
~~D. E. MORRIS~~
P. L. LAMAN

This Newsletter No. GN26-0887
Date December 3, 1976

Base Publication No. GC28-6394-4, -5, -6
File No. S360-24

Previous Newsletters GN26-0801 (-4,-5)
GN28-1047 (-4)
GN28-1062 (-4)

IBM DOS Full American National Standard COBOL

© IBM Corp. 1968, 1969, 1970, 1971, 1972, 1973

This technical newsletter, a part of Version 2 of IBM DOS Full American National Standard COBOL for Release 26 of DOS, of Release 3 of Version 3 of the IBM DOS Full American National Standard COBOL Compiler and Library, and of Release 2 of the IBM DOS/VS COBOL Compiler and Library, provides replacement pages of the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered.

Pages to be added and/or replaced are listed below:

Cover, edition notice
Summary of Amendments (#10 added)
xxxvi.i
37, 38
41, 42
49, 50
67, 68
87-90
143, 144
211-214
217, 218
223, 224
239-242
273, 274
293, 294
317, 318

Please place the Summary of Amendments page following the cover page.

Each technical change is marked by a vertical line to the left of the change.

Note: Please file this cover letter at the back of the publication to provide a record of change.

